PHD

An agent-independent task learning framework

Wood, Mark

Award date:
2008

Awarding institution:
University of Bath

An Agent-Independent Task Learning Framework

submitted by

Mark Alistair Wood

for the degree of Doctor of Philosophy

of the

University of Bath

Department of Computer Science

July 2008

Acknowledgements

My greatest debt of professional thanks is to my supervisor, Joanna Bryson. She has been completely invaluable in offering advice, guidance, encouragement, and inspiration when I've needed it most. By all accounts it is rare to find a supervisor who always answers e-mail; who is always willing to meet up and discuss your latest ideas; and who always looks for the best in every situation. I therefore count myself as blessed. One specific note of thanks is for motivating me to publish my work, as I never would have had the confidence to do so otherwise.

I'd also like to thank Will Lowe for various discussions and proof-readings throughout the course of my PhD, and for struggling against the odds to steer me onto the mathematical straight and narrow. Without Tristan Caulfield's programming expertise, the last phase of experiments simply could not have happened, so I thank him for this significant contribution. The opportunities I have had to turn my ideas over with Jan Drugowitsch and Hagen Lehmann have been without exception a beneficial pleasure. I should also mention my second supervisor, Nicolai Vorobjov, for supporting me at critical times, and Andrew Fitzgibbon who, although he almost certainly does not realise it, convinced me to continue at the half way stage. My examiners, Mark Humphrys and Daniel Richardson, provided much helpful feedback and many recommendations for improvement, and the dissertation is all the better for it. My thanks go to them both, and to Dan in particular, who helped me in ways far beyond the call of duty throughout my period of corrections.

My greatest debt of personal thanks is to my wife, Rebecca Wood. Her inexhaustible love, support, patience, and forbearance has meant everything to me, and without it I would have given up long ago. She has borne the brunt of the bad times, both emotional and financial, and has continued to stand by me unconditionally. Toward completion she even insisted on being my data entry clerk, which again made the final run of experiments possible. I love her very much, and thanks is a woefully inadequate word for all that she has done.

My parents, Robert and Dorothy Wood, must get an honourable mention at this point. They have always upheld me in every decision I've made in life, and

their influence during these PhD years cannot be underestimated. Thanks to Adam Dziedzic, who for me turned the lab into somewhere I wanted to go, and also his partners in crime: Emmanuel Tanguy, Sirapat 'Zero' Boonkrong, and Richard McKinley. These, my friends and colleagues, have shared the journey with me and it has been all the better for it. Life on campus would not have been the same without the many and varied friends I made through Bath University Christian Union; thanks to all of you for making an old man feel welcome. Last but not least, I'd like to thank the *brother john / Risen* brothers; Tom Maynard, Ian Wood, James Truett, Nathan Maynard, Tom Roe, Matt Roe, Tim Cracknell, Charles Webb, and Ben Taylor; for the prayers, for the fun times, and for keeping the faith.

Above all else, my thanks and praise go to Jesus Christ, my full-time Saviour and part-time Lord. Anything good that comes of this work is for Him, and through Him, and to Him.

Summary

We propose that for all situated agents, the process of task learning has many elements in common. A better understanding of these elements would be beneficial to both engineers attempting to design new agents for task learning and completion, and also to scientists seeking to better understand natural task learning. Therefore, this dissertation sets out our characterisation of agent-independent task learning, and explores its grounding in nature and utility in practise.

We achieve this chiefly through the construction and demonstration of two novel task learning systems. Cross-Channel Observation and Imitation Learning (COIL; Wood and Bryson, 2007a,b) is our adaptation of Deb Roy's Cross-Channel Early Lexical Learning System (CELL; Roy, 1999; Roy and Pentland, 2002) for agent-independent task learning by imitation. The General Task Learning Framework (GTLF) is built upon many of the principles learned through the development of COIL, and can additionally facilitate multi-modal, lifelong learning of complex skills and skill hierarchies. Both systems are validated through experiments conducted in the virtual reality-style game domain of *Unreal Tournament* (Digital Extremes, 1999). By applying agent-independent learning processes to virtual agents of this kind, we hope that researchers will be more inclined to consider them on a par with robots as tools for learning research.

Contents

List of Figures

List of Tables

Part I

Background and Model

Chapter 1

Introduction

Life is not always predictable. To survive in an uncertain world, all but the simplest living creatures need skills; patterns of behaviour which can be adapted to solve a given problem in a variety of contexts. In fact, this is not just desirable for natural agents, but for any that we would create to operate in uncertain domains. In environments as dynamic and varied as the Real World, evolution is rarely a sufficiently rapidly adaptive mechanism to hard-wire every specific skill necessary for survival. However, learning itself can be seen as a skill; a solution to the problem of acquiring the knowledge necessary to alter one's behaviour based on past experience. If evolution is able to provide this skill, then it can be used to acquire missing low-level skills. Another way of acquiring the skill of learning that is available only to created agents, is through design.

Bearing this in mind, it is important that we do not underestimate the importance of an agent's inherent biases. The fewer innate skills available, the larger the search space will be. Consequently it becomes less likely that the agent will learn some necessary but absent skill, and the probability of survival is reduced. However, some agents 'cheat' this trade-off by narrowing the search space in other ways, enabling them to learn more and thus retain maximal adaptability. For example, humans use their culture to gain access to tried and tested skills which have not been encoded genetically, but would be (too) expensive to learn individually.

1.1 Thesis in Brief

This dissertation explores a method of acquiring low-level skills which can make best use of both individual and social information channels; namely *task learning*. Even life itself can be seen as a task with the goals of survival and reproduction; goals which can be decomposed into many other sub-goals and sub-tasks. In this way, skills and tasks have a symbiotic relationship: tasks require skills in order to be completed, and (many) skills require tasks in order to be acquired and honed.

Studies in nature of how and why agent A learns task T, and engineering projects which attempt to endow agent A with the ability to learn task T, have been carried out for very many pairs (A, T). Often, the two research streams have informed and influenced each other. Indeed, this project began with our attempt to create agents called *bots* which could learn how to play the virtual reality-style computer game *Unreal Tournament* (Digital Extremes, 1999); one such (A, T). However, our experiences inspired us to ask a different question: how can agent A learn task T for *any* (A, T) possible in practise. In other words, what elements of task learning depend neither on the specific task being learned, nor on the agent doing the learning? The crucial follow-up question must be: what benefits could such a study bring to specific science and / or engineering projects?

1.1.1 Finding a General Description

At the core of this problem is the question of what it means to be an agent. In recent years, the word has come to refer to a broad range of entities, including some which are sufficiently abstract that 'task learning' has no meaning in their domain. We therefore make the following restrictions:

1. Agents are situated in space and time.

   We do not limit this to material or even Cartesian space, nor need time be continuous. Fundamentally, the agent and the environment it inhabits must be distinguishable.

2. Agents perceive.

   This includes both elements of the environment (exteroception) and its own state (proprioception).

3. Agents act.

   This includes both acting upon elements of the environment (actuation) and upon its own state (cognition). Actions take time.

4. Agents are resource-bounded.

   Their perceptual resolution, memory and processing speed are all finite.

So, far from being *tabula rasa*, our notion of agent-independence brings with it significant constraints within which to work. Humans clearly fit the description, as in fact do all animals and autonomous robots. Situated autonomous software (or 'virtual') agents are less obviously included, and one of the subsidiary aims of this dissertation is to elevate the status of virtual agent research by working within this common description framework, thus drawing it closer to its material counterparts. We are not proposing to design a complete agent architecture; we focus on task learning and skill acquisition, along with any necessary interfaces with the agent's other sub-systems. This frees us to create both a task-oriented description of our agent, and an agent-oriented description of tasks.

A Task-Oriented Agent Description

Tasks could conceivably contain many different types of elements and goals that once perceived may need to be processed in various ways. Our framework therefore uses grounded *perceptual classes*, capable of representing anything from low-level sensor readings to high-level concepts, providing that there is always a mapping back to the raw sensory input. Similarly, *action elements* can encode anything from single motor commands to co-ordinated sequences of movement[1]. An acquired skill or *behaviour* can then be stored as a function mapping perceptual classes to action elements, and the same representations can be used at (m)any level(s) of detail.

An Agent-Oriented Task Description

On the other hand, since tasks themselves are defined by goals, and goals must be defined by agents, it seems reasonable for tasks also to be described in terms

[1]Bearing in mind that 'motor' and 'movement' might not have particularly natural analogues for some of the agents under consideration.

of these 'agent-centric' representations. Practically, we associate task goals with a set of perceptual classes which must concurrently apply (a *perceptual state*) for the goal to be satisfied. Now, this does *not* mean that the agent learning the task must have knowledge of the goal states, or even a perceptual configuration that is capable of distinguishing them. The task goals could be set or defined by another agent, and would therefore be defined in terms of that agent's perceptual configuration. In any case, the *observer* is key, whether that refers to the agent itself; motivating, observing and assessing its own task performance; or whether it refers to an external assessor (or both).

1.1.2 Putting the Framework to Use

The remainder of this dissertation is primarily concerned with detailing and supporting the framework proposed above: a General Task Learning Framework (GTLF). We now put forward some ways in which, assuming such a construct is at least conceivable, it might be useful.

Engineering

First, let us consider the potential benefits to engineers attempting to build agents capable of task learning. GTLF provides universal representations for perception and action, as well as specifying the flow of information between its component modules. The constraints of these representations and processes should be flexible enough for use by designers favouring symbolic (i.e. GOFAI[2]), non-symbolic, and hybrid AI systems. The modules themselves are underspecified; some are also optional, and for those that are required, we give some example default procedures in Chapter 3. This allows an agent designer to implement only those modules relevant to their particular research problem, as well as conferring all the usual benefits of the modular programming approach (code transfer / re-use, parallel development, ease of debugging, etc.; Sommerville, 2006). If it is an individual component rather than a whole agent system that is of interest, GTLF also in principle provides a platform for comparison of behaviour models, learning algorithms, performance comparison metrics, and so on (although we do not demonstrate this practically within this body of work). The agnostic

[2]Good Old-Fashioned Artificial Intelligence.

nature of the learning system to its implementation domain should allow for a wide sphere of application, as well as easing domain transfer; for example, from simulation to robot platforms, as is a common process.

Science

There are also a number of potential benefits from the perspective of scientific investigation. Firstly, we hope GTLF will help researchers (in both biology and technology) consider the task learning process in isolation, without being unduly influenced by a particular case-in-point. Then when returning to concrete examples, it should be easier to identify common features and requirements of the process, for example. Also, although the system as a whole is not a model of task learning in any particular species, the individual components are to some extent biomimetic (see Chapter 2). It can thus be used as a starting point for more biologically plausible models of task learning; a rough sketch to be refined and to stimulate thinking. We believe that one of the most interesting avenues of study using GTLF is the role of social learning, and how it interacts with and complements individual learning, and we show how this could be done in Chapter 7. Finally, as we mentioned above, by designing a framework which applies to many different classes of agent, and demonstrating its use for virtual agents, we hope to encourage others to use virtual agents as a research tool (Laird and van Lent, 2001).

1.2 Contributions

We view the primary contribution of this thesis to be:

- The **General Task Learning Framework** (GTLF): for incorporating and investigating task learning in situated agents. Our version of GTLF has been implemented as a package of Java classes, and can be integrated with any agent which accepts string input / output.

Additionally, we view the secondary contributions to be:

- **Cross-Channel Observation and Imitation Learning** (COIL): an adaptation of Deb Roy's Cross-Channel Early Lexical Learning system (Roy,

1999; Roy and Pentland, 2002) to learning by imitation. Also implemented as a Java package.

- A formal method for defining task performance metrics, inspired by the work of Nehaniv and Dautenhahn (1998, 2001).

- An extension of the JavaBots package (Marshall, 2000; Adobbati et al., 2001) for interfacing Unreal Tournament bots with external AI programs. Our additional classes improve the interface with our learning systems (and in principle with others), as well as including useful game object representations and helper functions.

1.3 Supporting Publications

Chapters 5 and 6 of this dissertation are supported by:

Wood, M. A. and Bryson, J. J. (2007). Skill Acquisition through Program-Level Imitation in a Real-Time Domain. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(2):272–285.

Chapter 6 is also supported by:

Wood, M. A. and Bryson, J. J. (2007). Representations for Action Selection Learning from Real-Time Observation of Task Experts. In Veloso, M. M., editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI '07)*, volume 1, pages 641–646, Hyderabad, India. IJCAI, AAAI Press.

Chapter 2 is supported by:

Bryson, J. J. and Wood, M. A. (2005). Learning discretely: Behaviour and organisation in social learning. In *Proceedings of the Third International Symposium on Imitation in Animals and Artifacts*, pages 30–37, University of Hertfordshire, Hatfield, UK. SSAISB, AISB.

Related work on task learning not described in this dissertation includes:

Wood, M. A., Leong, J. C. S., and Bryson, J. J. (2004). ACT-R is almost a model of primate task learning: Experiments in modelling transitive inference. In Forbus, K., Gentner, D., and Regier, T., editors, *Proceedings of the Twenty-Sixth Annual Conference of the Cognitive Science Society*, pages 1470 – 1475, Chicago, Illinois, USA. Cognitive Science Society, Lawrence Erlbaum Associates, Inc.

1.4 Road Map

Following this introduction, Chapter 2 introduces our agent-independent description of task learning, explaining the basic principles of the framework with reference to biological and technological cases of social and individual learning. Chapter 3 then presents GTLF in full detail, examining each module in turn alongside hypothetical implementation examples. To conclude the first part, Chapter 4 steps back from this detail and attempts to uncover the broader design principles behind GTLF, and how they might best be applied by an agent designer.

The second part of the dissertation shows how we arrived at GTLF through a chronological account of systems development. The story starts with CELL, a system which we identified as having successfully achieved task learning in the particular context of lexical acquisition. Chapter 5 describes our adaptation of this system, which we called COIL, to learning by imitation in general. COIL's operation is demonstrated through learning two tasks by imitation in the virtual 3D domain of the 'First Person Shooter' game, *Unreal Tournament* (Digital Extremes, 1999). In fact, all of the experiments reported in this dissertation are conducted using this real-time, non-Markov testbed. Chapter 6 looks further into what was learned from COIL, in particular identifying its limitations. We then show how COIL can be improved using alternative learning algorithms, and use this principle to justify the design of the more general framework: GTLF. The particular extension to COIL that we implement here uses *Multi-Layer Perceptron* (MLP) learning, but this is just an example of one possible (class of) learning algorithm(s), as opposed to being integral to the path we trace:

$$\text{COIL} \longrightarrow \text{'Extended COIL'} \longrightarrow \text{GTLF}.$$

In contrast with COIL (and its extension), which focuses solely on imitation

learning, GTLF also encompasses *insight*, *trial-and-error*, and *instruction*, although only imitation and trial-and-error (in this case, Reinforcement Learning) have been implemented and tested thus far. We report this in the first part of Chapter 7, followed by results (again, using Unreal Tournament agents) demonstrating how GTLF could be used to investigate the interplay between social and individual learning. The second half of the chapter compares and contrasts GTLF with its nearest-neighbour learning systems in the literature. These include the robot task learning system of Nicolescu and Matarić (2001, 2002, 2003, 2007); Dogged Learning (Grollman and Jenkins, 2007); the virtual pilot system of Sammut et al. (1992); Morales (2003); Morales and Sammut (2004); and learning by imitation in Quake II (id Software, 1997; Bauckhage et al., 2003; Thurau et al., 2004a,b,c, 2005; Bauckhage and Thurau, 2004; Gorman et al., 2006a,b; Gorman and Humphrys, 2005, 2007). Although many of the key ideas behind GTLF were inspired by other systems (eg. Roy, 1999; Bryson, 2001; Nehaniv and Dautenhahn, 2001), the framework itself is our work alone. We have not yet had the opportunity to collaborate, and as yet (as far as we know) no other research groups have made use of it. However, all of the code related to our experiments has been written in Java and is available online[3], and although these classes have thus far only been applied to the problems described in this dissertation, it is our hope that others will download and make use of them.

The final part begins by exploring how GTLF could be used in current research applications, such as robotics, and also how it could be integrated with and supported by other learning systems. Chapter 9 moves from the practical to the philosophical, and explores the wider implications of GTLF to embodied intelligence and cognition. We end with an overview given the context of the entire thesis, ideas for fruitful future research problems, and some concluding remarks.

[3]Code available at `http://www.cs.bath.ac.uk/~cspmaw/disscode.zip`

Chapter 2

An Agent-Independent Theory of Task Learning

Many human skills are acquired through learning how to do a task and then practising it (Newell and Rosenbloom, 1981). The knowledge required for this can come from many sources, both external and internal. In this chapter, we further examine the representational substrate needed to describe task learning, and the process by which task learning results in skill acquisition and improvement. In doing so we look at four different learning methods which may be available to an agent. Although human task learning is the richest motivating example, we wish to make the discussion that follows as general as possible, making only the assumptions laid out in Chapter 1; i.e. that we are talking about *situated agents*:

> "An agent is said to be situated if it acquires information about its environment solely through its sensors in interaction with the environment. A situated agent interacts with the world on its own, without an intervening human[1]. It has the potential to acquire its own history, if equipped with appropriate learning mechanisms."
> (Pfeifer and Scheier, 1999, p. 656)

We assume nothing further about their form or capabilities, nor whether they are natural, robotic or virtual. We begin by detailing the building blocks motivated and introduced in Section 1.1.1, before moving onto the learning processes them-

[1]It is clear from the broader context that Pfeifer and Scheier are here referring to human teleoperation, as opposed to human interaction with the agent in the environment.

selves. As many of the terms we use are in common parlance, it is particularly important that we tie down our own interpretations.

2.1 Representations

Firstly, we define an agent's *sensor space* as the space of all possible readings from all sensors. This includes external stimuli, proprioceptive and other internal sensor data, and even working or long-term memory. Recall that our fundamental unit of perception is the **perceptual class**, which could correspond to any distinct subset of sensor space. Intuitively, a perceptual class defines some property (or set of properties) of an agent's sensor space; equally, some property (or set of properties) in sensor space could be used to define a perceptual class. We refer to the set of mappings an agent makes from sensor space into perceptual space as the agent's **perceptual system**.

The above definition of perceptual classes could be seen as an agent-independent abstraction of *perceptual symbols* (or *images*) as described by Barsalou (1999). In that paper he develops a model of human cognition, integrated with both the neuroscience and psychology literatures, which uses symbols that remain 'grounded' in the neural substrate. He proposes, for example, that the neural pattern generated when perceiving an object is symbolised by a compressed neural pattern which represents a concept of that object.

Perceptual classes are assumed to remain similarly grounded in 'sensory-motor space', but that space may not be neural — that depends upon the agent in question. For instance, the concept of sensory-motor space for a virtual agent requires an abstract definition of sensors and actuators (see also Sections 5.2.2 and 9.1). Apart from the theoretical correlates, categorisation also has clear practical benefits. Since many sensors perceive a continuum of values, categorisation helps to make decision-making tractable by decreasing the size of the state space (Ueda et al., 2004). On the other hand, there are also circumstances in which smoothness and continuity are preferable to discreteness (see Section 10.3).

The propriety of a given categorisation depends entirely upon the task at hand and the degree to which the agent has learned it. Perceptual classes can be nested (e.g. `car`, `vehicle`, `object`) and overlap (e.g. `red`, `car`, `red_car`). Given that tasks can be described using different goal granularities (see below),

perceptual class categorisations need similarly to be able to 'filter' the world at an appropriate level for decision-making. A *coarse* categorisation results in the agent making decisions based on broad, high-level classes, whereas a *fine* categorisation implies that classes are narrow and low-level. We refer to the set of all perceptual classes that an agent 'occupies' at a given time as the agent's **perceptual state**.

Any physical or mental activity which is motivated by a goal or sequence of (sub)goals can be described as a **task**. A **goal** (or *effect* — see Section 3.3.1) specifies a perceived state of the world (including the internal state of the agent) which must be reached in order for the goal to be satisfied. We argue that goals (and consequently tasks) are always generated and monitored by agents; therefore goal states correspond to perceptual states. Goal states can be generated from three possible perspectives, depending upon how the agent's progress is being assessed:

1. Another agent external to the environment; a passive observer such as the agent's designer, for example.

2. Another agent situated within the environment, possibly able to provide feedback to the agent; a conspecific agent, for example.

3. The learning agent itself.

If a task is being assessed from more than one viewpoint, the measures of progress may differ. For example, suppose an agent has set itself the task of building a wall. If, at the end, the wall remains upright, the agent may consider the task to have been successfully completed. However, bricklaying is a skill with a recognisable social standard attached to it, and simply remaining upright may not be sufficient to fulfil it. This is even more the case if the handiwork were being assessed by an expert builder. In contrast, the task of carving a sculpture has far fewer globally accepted constraints.

It is possible to describe a task not only from different perspectives, but also at many different levels of detail; that is, at different *granularities*. This concept is used by Nehaniv and Dautenhahn (2001) in the context of comparing two behaviours. At a coarse granularity, building a wall using bricks and mortar could have the same series of goals as building a wall using toy wooden blocks:

1. Choose a level surface.

2. Set down a row of bricks.

3. Set down another row of bricks on top of and offset with the row below.

4. Repeat (3) until the wall is high enough.

At this level they are effectively the same task: *building a wall*. At a finer granularity, however, the sub-goals could be very different (i.e. *apply mortar to trowel*). If we fix the granularity of description, then tasks can be put into equivalence classes, or *task classes*, such that member tasks have the same sequence of sub-goals. Referring back to the example above, *building a wall* is a task class which includes both building with bricks and building with blocks; at a coarse descriptive granularity the two tasks have the same sequence of sub-goals. At a finer granularity we could find another task class, *building a brick wall*. This class would contain members corresponding perhaps to different types, heights, shapes, etc. of brick wall since each of these tasks could share the same sequence of sub-goals. This class, however, would be distinct from the *building a block wall* class, which could similarly contain members representing different block walls, but would have a different sequence of sub-goals to the brick wall class. We further discuss granularity and its relation to measures of task error in Section 3.3.1.

An **action element** can refer to anything that is executable by an agent as a single unit, ranging from, say, low-level motor commands to learned high-level movements. Like the perceptual class, it is thus capable of covering many levels of task description. We borrow the term from Bryson (2003), where it is similarly used to describe both primitive and aggregate actions. For example, when learning to pick up a ball, it may be that the only action elements needed are individual motor commands, e.g. `extend_thumb()`, `extend_first_finger()`, etc. Learning to juggle, however, requires action elements which are composed of other action elements, such as `pick_up_ball()`, `throw_ball()` and `catch_ball()`. Learning to put on a circus show requires action elements such as `juggle()`, `walk_tightrope()`, `tame_lion()`; and so on.

A **skill** is a specification of transferable behaviour which when executed in a suitable environment can bring about the completion of a task or tasks. It is represented as a function

$$s : \mathcal{P}(P) \rightarrow A \tag{2.1}$$

where P is the set of perceptual classes discernible by the agent, $\mathcal{P}(P)$ is the power set[2] of P, and A is a subset of the action elements possessed by the agent (see Section A.1 for more details). It is analogous to a *policy* as described in the Reinforcement Learning and Operations Research literatures (Sutton and Barto, 1998; Hillier, 2004), and to a *behaviour* in the Behaviour-Based Artificial Intelligence (BBAI) and related literature (Matarić, 1997; Arkin, 1998). We will often adopt particularly the latter term in reference to skills.

From the above definitions, we see that an action element may itself be a skill. This allows for a hierarchical skill structure in which high-level skills (e.g. `put_on_circus_show()`) can map coarse-grained perceptual classes (e.g. `lion_is_attacking`) to other lower-level skills (e.g. `tame_lion()`). For a skill to qualify as an action element and therefore be available to other skills, it must be executable as a unit; in other words, it must either be innate or both acquired *and* mastered. Skills can be acquired and improved by learning and practising tasks which require those skills, through the alteration of the perception-action map. Skill improvement with respect to a task class can be measured by an increase in *accuracy* (fewer mistakes) and / or *efficiency* (less time, memory or steps required). However, improving a skill by learning only one task from a class does not guarantee improvement for the other member tasks. Indeed, the skill may become *overspecialised* and performance on other tasks may decrease (Ratnieks and Anderson, 1999). Therefore, a skill is improved in general if its performance increases on average across all tasks in a given class. We say that a skill is *complete* with respect to a task if every perceptual state that can be entered in the task environment is associated with an action (also see Section A.1). In other words, however inaccurate its actions may be, the agent always has *something* to do.

Now we have our definitions in place, one more thing to consider before the task learning process itself is the prior state of the agent.

2.2 Prior Knowledge

At the point at which task learning commences, an agent will possess a certain amount of knowledge. This could have been accumulated genetically via evolu-

[2]That is, the set of all subsets.

tion, gifted by the agent's designer, acquired through previous interactions with its environment, or a combination of these (see Chapter 1). We wish to better understand this *prior knowledge*, and how it relates to any new task knowledge the agent acquires. Prior knowledge can be categorised as follows:

2.2.1 Innate Skills and Biases

Innate skills and biases refers to the abilities afforded, and limitations and tendencies imposed, by an agent's composition. Firstly, agents will be equipped with a finite number of sensors, which each operate with only limited scope and precision. This imposes natural limits upon the perceptual classes that can be formed. Similarly, the agent's structure and motor capabilities limit the number of available actions. In addition to these absolute limitations, the agent could have certain 'hard-wired' tendencies which serve to ease the complexity of the learning problem. It may be that some high-level perceptual classes 'come as standard' even though the agent is capable of perceiving at a finer grain. The same is true for nontrivial action elements, or even whole behaviours which are reflexive or automatic in some way. Apart from these perceptual-motor biases, there are other more abstract mental capabilities that can strongly influence the learning process. Indeed, learning itself can be done in many ways (see Section 2.3.1), some of which will be more natural for a given agent than others. This in turn may depend on other factors such as the richness of mental representation available; memory capacity and accuracy; ability to reason, predict and abstract; and so on. For example, human infants are born with relatively few skills[3], but have advanced capacities and strong biases for learning through various means. Humans may have a longer maturation period, but eventually master a wider variety of skills when compared to other animals born with more skills, but with lesser learning capabilities and having shorter maturation periods (Sloman and Chappell, 2005).

[3]And there is evidence to suggest that learning takes place even before birth (Mennella et al., 2001).

2.2.2 Acquired Skills and Biases

Acquired skills and biases encompasses anything that the agent has learned during its lifetime through interaction with the world it inhabits. For example, some actions may have proved to be generally easier, more rewarding or more frequently observed than others. Equally, some may have accumulated negative connotations or even proved redundant. Some perceptual classes, like those associated with danger or sustenance, may have high prior saliency compared to more common percepts. Prior task learning episodes may have necessitated the creation of new compound actions which still reside in memory, and are available for application in new tasks. They may also affect the granularity of perception with which the agent approaches new tasks. For example, suppose some previous task required a high level of scrutiny in a particular region of perception space. The agent may then start a new task with a similarly fine-grained categorisation in that region. Conversely, a task requiring only broad categories to enable correct behaviour could bias the agent toward using a coarser perceptual granularity. Not only such perception and action configuration, but also general associations (or dissociations) between them can be acquired through experience. Certain actions in certain situations may have always yielded reward or punishment, for instance. A highly experienced agent may have many stored skills when coming to a new task, and these will inevitably affect the learning primitives and paths chosen.

Having considered the knowledge an agent brings to a task, we now define the task learning process itself in more detail.

2.3 The Task Learning Process

We propose that the task learning problem comprises four major sub-problems:

1. *Forming or reinforcing associations* between attended perceptual classes and known action elements.

2. *Improving the selection of perceptual classes to attend to* in the case that many concurrently apply (Wood et al., 2004).

3. *Creating new perceptual classes* to allow for finer-level behavioural distinctions where mistakes are being made.

4. *Forming new action elements* where behaviour is proficient, to allow for more efficient organisation.

The first of these is equivalent to the forming of new skills, and the improvement of existing ones, and later in this section we look at the various sources of knowledge available to this process. The second problem of learning a strategy for attention, and the reasons why this may be necessary in the first place, are then covered in Section 2.3.2. By iterating between these two processes, an agent with good prior knowledge may be able to acquire accurate and efficient skills. However, if prior knowledge is poor, the agent's perception (3.) and action (4.) representations may need to be reconfigured to allow for continued improvement. This is the subject of Section 2.3.3.

2.3.1 Forming and Improving Skills

To isolate this part of the process, let us assume that our agent has a fixed perceptual categorisation, a fixed action repertoire, and some predetermined way of choosing which perceptual classes to attend to and which to ignore (if any) at a given time. We identify four different learning methods by which an agent may be able to discover perception-action associations:

Insight

Insight refers to the process of forming hypotheses for novel behaviour using prior knowledge and mental processes *only*. That is not to say that the perceptual *input* must be of a certain type (i.e. memory, proprioception, etc.). We have already established that the perceptual system makes no explicit distinction between different sources, and in any case, insight learning can make use of exteroceptive input. It is the absence of social input or interaction with the task, and hence the reliance on mental processing, that defines this learning method.

For example, suppose that the task is to retrieve some food that is out of the agent's reach, and that there are various boxes and poles available in the environment which the agent could make use of. If the agent is able to complete the task

through appropriate manipulation of the available tools *without* previous experience of the same task, the chance to experiment with the tools, the opportunity to observe the task being completed, or any kind of external guidance, then we would describe this as learning through insight. For this to be possible, the agent would at least need some relevant knowledge or experience of some elements of the task; e.g. that boxes can be stacked and stood upon, that poles can be used to extend reach, etc.

In general terms, insight requires some prior knowledge of the task domain. At the human-level, insight can make use of our cognitive skills for symbolic representation, reasoning, inference, deduction, abstraction, generalisation, and so forth. However, there are claims that animals too are capable of insight. Köhler records probably the most famous (and controversial) case, in which chimps apparently solve the very task described in the above example (Köhler, 1925). Even non-primates such as pigeons (Epstein et al., 1984) and crows (Chappell and Kacelnik, 2002), when faced with similar problems, seem to exhibit this ability. For other examples in biology, see a recent review of animal reasoning by Watanabe and Huber (2007).

Although insight by definition supposes no physical interaction with a task, it is strongly dependent upon trial-and-error learning (see below) for two reasons. Firstly, no matter how advanced the powers of insight at an agent's disposal, forming a complete solution to a complex task in an unpredictable environment is unlikely. Fine-tuning can come through trial-and-error. Secondly, a solution formed through insight may be entirely correct, but the only way the agent can verify this is through test interactions with the task (see Section 3.3).

Trial-and-error

In contrast with insight, learning by trial-and-error or *Reinforcement Learning* (RL) *requires* the agent to interact with the task environment. During this interaction, the environment will administer rewards and / or punishments which can be used by the agent to guide behaviour improvement.

This time let us suppose that the task is to play a certain tune on a piano. We will assume that the agent knows the target tune, and that pressing piano keys causes notes to be played, but does not know the key-note correspondence. Providing that the agent has an aural sensor and a metric for measuring the

distance between heard notes, the correct tune could be learned by pressing a random key and then improving its choice next time. The pitch difference between the target note and the played note constitutes a 'punishment' which the agent can attempt to minimise. If the agent has no aural sensor (is deaf) or no such metric (is tone-deaf), and has no other means of receiving punishment (by vibration, for example), the task becomes impossible using this method alone.

In principle, any agent that can perceive and 'interpret' environmental feedback can utilise reinforcement learning. These minimal requirements should make it more widely available than the other learning methods. However, for RL to be tractable, the task itself needs to be sufficiently constrained, or equivalently, an agent's prior biases need to be sufficiently strong. For example, very simple biological and artificial agents are able to use RL to solve equally simple tasks. Worms can learn to approach or avoid tastes, odours or temperatures that predict the presence or absence of food (Rankin, 2004). Software agents that catalogue internet pages, 'web spiders', can learn to choose better hyperlinks as they crawl (Rennie and McCallum, 1999). When it comes to more complex tasks, however, agents may not possess an adequately rich representational substrate, or may not be able to commit the time and resources required to solve them. A task behaviour (or *policy*) can improve only slightly with each interaction, and as the required behaviour becomes more complex it becomes hard and time-consuming (if not intractable) to acquire it using trial-and-error alone (Littman et al., 1995). Also, some situations yield irreversible or crippling punishments (death being the most extreme example) if certain actions are taken. Clearly, a total reliance on reinforcement learning in these circumstances is best avoided.

Observation

An agent learning a task may also be able to gain relevant information by *observing* physical interactions with the task environment, rather than by taking part in them. We distinguish between three different types of learning by observation:

1. *Nonsocial learning* — where the task interactions are caused by physical effects in the environment. This includes 'natural' effects such as gravity and wind, and 'mechanical' effects such as automated signals and switches.

2. *Nonimitative social learning* — where the task interactions are carried out

by other agents, but not as part of an attempt to complete the task.

3. *Imitation* — where another agent can be observed while attempting to complete the task.

All of these forms of learning are united by their reliance on at least a partial solution to the *correspondence problem* (Nehaniv and Dautenhahn, 2002). That is, agents must possess a *correspondence library* linking *allocentric* and *egocentric*[4] actions, states and / or effects (depending on the task description and the learning methods available).

If our piano-learning agent could observe a player piano[5] playing the piece of interest, then this would constitute nonsocial learning. If a human pianist were to sit and play a scale instead of the piece, task-relevant knowledge could still be obtained from the key-presses — this is nonimitative social learning. Lastly, if a piano maestro were to perform the piece in a recital, learning would be by imitation. Note that for all of these examples, the observer could be entirely incidental to the interactions.

Learning by observation seems to come naturally to humans, with infants being capable of facial and gestural imitation near birth (Meltzoff and Moore, 1977), and goal-oriented imitation within a few years (Bekkering et al., 2000). As for other animals, the most widely cited example is that of the cultural transmission of sweet potato washing behaviour in Japanese macaques (Imanishi, 1957). Parrots (Pepperberg, 1994) and dolphins (Herman, 2002) are also apparently capable of a kind of goal-oriented imitation which goes beyond simple mimicry. In the ethology literature, social influence, social learning and imitation have been deconstructed into many different sub-categories (Whiten and Ham, 1992; Zentall, 2001). Since we are concerned with task learning, that is, learning how to satisfy a sequence of goals / achieve a sequence of effects, we use a very broad definition of imitation that captures this level of behaviour replication called *effect-level imitation* (Nehaniv and Dautenhahn, 2001). We discuss this in Section 3.3.1, in relation to measuring task performance.

[4]We use *allocentric* to mean 'from an external perspective', and *egocentric* in place of 'from the perspective of the observing agent'.

[5]A type of piano developed in the late 19th century which uses mechanical, pneumatic or electrical devices to strike the keys in place of a human player.

Instruction

Learning by instruction, like learning by observation, makes use of a source of knowledge external to the agent. Instruction differs, however, in that it involves the explicit communication of lexical symbols, in contrast with the symbols generated internally via the correspondence library during learning by observation. The most obvious examples are spoken or written language, and gestural signals. To take advantage of this learning method, an agent must be able to obtain instruction from a suitably qualified teacher agent. Also, these instructions must be communicable and interpretable, which in turn requires teacher and student to share some kind of common lexicon.

Assume now that the tune is *unknown* to our piano-learning agent, but that a score has been provided to read from. As long as the agent knows how to translate the written notes into keystroke instructions, then it can use them to generate behaviour in tandem with its piano-playing skill. The instructions themselves are provided by another agent (in this case indirectly) that has knowledge of the task. They are given for the specific purpose to convey information which reduces learning time and raises the saliency of the relevant elements of the task environment.

Caro and Hauser (1992) give two broad categories of teaching in nonhuman animals:

> "... situations where offspring are provided with opportunities to practice skills ("opportunity teaching"), and instances where the behaviour of young is either encouraged or punished by adults ("coaching")."

We, on the other hand, would see both of these as examples of learning by a combination of observation and trial-and-error. Observation in the first case, because salient elements of the task are brought to the attention of the young by the adult through interaction, which requires a correspondence library. Trial-and-error in the second, because the encouragement and punishment provided by the adult mirrors the reward function required for reinforcement learning. While we fully support Hauser's definition of teaching, and would indeed be happy to use it ourselves if we were studying teaching in isolation, our narrower definition better suits our purposes. We are considering a more general class of agent than Hauser (with a view to constructing our own), therefore the distinction we make

between learning methods is determined to a greater extent by their *requirements*. This is not to say, though, that humans are the only animals capable of symbolic (or *functionally referential*) communication. Ants (Hölldobler, 1999) and bees (Gould, 1975), for example, use chemical and movement cues to refer to locations outside of their nests.

Summary

In addition to the different requirements for each method set out above, they also have some in common. For example, learning from past experience is not possible (by definition) if an agent has no capacity for storing state (memory). Also, effective learning by observation and trial-and-error relies on at least rudimentary statistical inference[6].

Different contexts and task classes suit different learning methods, and an agent that is able to take advantage of this by switching between them as appropriate should do so, or preferably use all of them in parallel and then consolidate the knowledge centrally for maximum gain. Instruction, imitation and nonimitative social learning are only available to agents who have access to relevant social input for a given task. Nonsocial observation learning, although possible for nonsocial agents, still requires external input which may not be present. Trial-and-error can be used by any sufficiently capable agent, provided that learning is appropriately constrained, and the task is conducive to repetitive interaction. Caution should be exercised before using RL for time-critical and / or dangerous tasks, for example. Insight can in principle be used for the broadest variety of tasks, but makes the highest demands upon the mental capacities and prior knowledge of the agent.

2.3.2 The Attention Problem

An agent with a variety of skills will be able to discern many different perceptual classes. However, not all of these classes will be relevant to every task, even if many of them are present in the environment. We refer to the problem of choosing which perceptual classes to select when more than one concurrently applies as

[6]That is, some kind of recognition that the more frequently an association is observed, the more likely it is that the association holds in general.

the attention problem. But why is this necessary?

Recall that the perceptual system outputs the perceptual state (the set of all perceptual classes that currently apply) given the sensor state. Recall also that a skill is a map from subsets of this state to action elements. If there were no intermediate processing, then skills would be required to encode a map from *every* possible subset of the perceptual state. For simple, sequential tasks, highly constrained environments and / or carefully tailored perceptual representations, this may not be a cause for concern. However, an increase in task complexity necessitates a parallel increase in the complexity of the agent's perceptual configuration. This in turn causes an exponential increase in skill complexity[7] and therefore acquisition time. To avoid this problem, we implement a secondary stage of processing: an attention strategy. This serves to select a subset of the perceptual state to present to the learning methods (above), thus sharing the complexity load.

Input Selection vs. Action Selection

This trade-off between skill complexity and attention strategy complexity can be thought of in terms of a trade-off between *input selection* and *action selection*. To illustrate, let us first suppose we are dealing with a single, 'standalone' skill with an associated attention strategy. The attention strategy performs *input selection*; it narrows the input space by selecting some subset of the full perceptual state. The skill is a mapping from the input subspace (provided by the attention strategy) to action; the skill performs *action selection*.

Now let us consider the more complex (but also more common) case in which an agent possesses multiple skills for carrying out multiple tasks and achieving multiple, possibly conflicting, goals. Assuming each skill has an associated attention strategy, we know that any given skill will select an action for our agent (or possibly no action if that skill is not *complete* on the input space — see Section 2.1). The question becomes, then, how to select which *skill* to 'listen to' in the case of multiple tasks / goals being present in the environment. Back in Section 2.1 we argued that goals only exist in terms of agents and must always be

[7] As the maximum size of the perceptual state (i.e. the total number of discernible perceptual classes) $|P|$ grows, the number of subsets (i.e. the domain of the skill function) $|\mathcal{P}(P)| = 2^{|P|}$ grows exponentially.

represented in the perceptual state. Put another way, all the information needed (or at least all the information *available* to the agent) in order to select which skill should dictate action is *necessarily present* in the perceptual state. The remaining job of skill selection or *goal selection* is really just action selection at a higher level: given some input perceptual state, choose which *skill* (rather than which action) to execute.

In principle we see no reason why such goal / action selector hierarchies could not be several layers deep; we talk about this further in Section 3.5.1. At any given node of this hierarchy, the attention mechanism could be made to work hard and reduce the state space available to the goal / action selector, or the hard work could be done during the goal / action selection process itself.

Biological Attention Mechanisms

Attention is a common phenomenon in biology. We cite mainly examples from human studies, since as agents we are the most adept at managing perception and cognition to solve highly complex tasks. On this subject, Pashler (1998, p. 2) writes:

> "...the mind is continually assigning priority to some sensory information over others, and this selection process makes a profound difference for both conscious experience and behaviour."

This perceptual selection seems to be associated with two major factors. The first of these is the broad category of *resource constraints*, which can be further subdivided into *perceptual* and *cognitive* constraints. Perceptual constraints include those limitations imposed by an agent's sensory capabilities. For example, the human retina has high resolution only over a few degrees of visual angle, limiting the number of items to which we can simultaneously attend (Pylyshyn and Storm, 1988). Our use of *space-variant active vision* (also used by all other higher vertebrates) also caters for our cognitive constraints. Indeed, Schwartz et al. (1995) claim that if our brains had to handle image representation at full resolution across our entire field of view, then they would need to weigh many thousands of pounds. This enormous space-complexity is mirrored by the NP-Complete computational complexity of unbounded visual search (Tsotsos, 1990).

The human attentional system thus makes use of innate genetic biases to introduce sufficient bounds on this complexity.

The second reason we should attend, particularly when it comes to task learning, is that not all stimuli are equal. Some may be irrelevant; their status bears no relation to the task at hand. Some may be unreliable; they are bad predictors of task-critical relationships. Some may be distracting, actually interfering with more useful stimuli. It makes information-theoretic sense to filter out 'bad' stimuli such as these, and focus on those which provide maximal predictive power (Dayan et al., 2000).

So, given that attention confers a number of advantages to a learner, how can a suitable strategy be learned?

Learning an Attention Strategy

The problem of learning a strategy for a given (i.e. fixed) perceptual categorisation is interdependent with the problem of optimising the categorisation itself with respect to a task. The latter is complex in its own right, and is addressed in Section 2.3.3 below. We first look at how the six sources of task-related information listed in Sections 2.2 and 2.3.1 could contribute to selecting from a known configuration:

Innate Bias Some perceptual classes, such as those signalling danger, may have a pre-programmed high priority. Innate prioritisations may or may not be able to be modified by subsequent learning; for example, the re-training of phobic behaviour (Öhman and Mineka, 2001).

Acquired Bias Past interactions with the world may have proved some perceptual classes generally more salient than others. This knowledge can be passed into a new task, introducing a bias for attention (which could be beneficial or otherwise).

Insight The agent's reasoning system may allow it to infer at least an initial strategy estimate before any external interaction is necessary.

Instruction If provided with instructions, these may explicitly or implicitly contain priority information.

Observation Suppose that an agent has learned a skill which maps the set of perceptual classes P onto action a. Suppose the agent then observes another agent exercising this skill execute action a when some of the classes in P are missing from the perceptual state. This allows the observer to hypothesise a new, tighter attention strategy.

Trial-and-error To infer a strategy by trial-and-error, the relative reward of attending to some sets of classes above others (inasmuch as they may dictate different actions in the same perceptual state) must be sampled.

2.3.3 Iterative Episodic Learning

So far we have suggested methods for learning perception-action associations, and for learning a good attention strategy, which should both lead to better task performance. It may be, however, that these processes hit a ceiling, unable to further improve behaviour because the agent's task representation itself is not adequate for a solution. Now, attempting to learn associations and an attention strategy while simultaneously altering the very components referenced by these processes seems unwise if not impossible. If, however, the learning process is interrupted (either naturally or artificially), then any necessary reconfiguration can take place during these respites. We refer to such discrete learning sessions as **episodes**. The remainder of this chapter looks at the reconfiguration process and the sources of information which can be used to inform it. Before moving onto this, however, it is worth noting two further advantages to so-called *episodic learning*. Firstly, it allows for optional batch processing of data acquired during an episode, potentially enabling the use of more resource-intensive learning algorithms and / or relieving the pressure on other sub-systems. Secondly, it gives agents the opportunity to move between different tasks in a task class, lessening the likelihood of overspecialisation.

The Learning Loop

Suppose that all the knowledge gained during a given learning episode is accumulated in a buffer called the *episodic buffer*. This is our agent-independent adaptation of the episodic buffer that Baddeley conjectures to be a component of human working memory (Baddeley, 2000, 2001). It is similarly "assumed to

be a limited-capacity temporary storage system that is capable of integrating information from a variety of sources". At the end of each episode, the knowledge in the buffer is then integrated with the skill being learned. In this way we can define a 'learning loop' which iterates with each learning episode. Skill performance should show an overall trend of improvement as the loop iterates, with the process terminating when the improvement rate falls below a certain threshold, when 'optimal' behaviour has been learned, or after a predetermined timeout (effectively, when the agent gets bored). As was said in Section 2.2, improvement can be gauged in terms of increase in *accuracy* and increase in *efficiency*, but we need to establish metrics on both of these factors to make this possible in practise.

We have loosely defined accuracy as a measure of the number of mistakes made when completing a task, where perfect accuracy implies faultless execution. However, task goals are set by an observer (see Section 2.1), and deviations from these goals can only be defined by that observer. In other words, each task assessor must also supply an accuracy error metric. Even if a task is assessed to have been completed perfectly on a given attempt, it may be that errors in behaviour exist in an area of the task space not entered during that attempt. But if the error metric is defined on the whole of task space, and the agent's behaviour can be accessed directly, then a more accurate assessment can be made than would be possible using empirical trials. Also, if the metric takes into account not just the specific task being learned, but the entire task class, then overfitting (discussed below) can be alleviated.

Suppose a perfectly accurate skill has been acquired, resulting in zero error across a task class. It may still be that the amount of memory, number of decisions, perceptual scope, etc. required by the skill causes problems for an agent with limited resources. The impact of these factors is dictated by the structure and granularity of perception space (see Section 2.3.2), as well as the complexity of the actions available in the agent's repertoire. These can be compared numerically in terms of their respective building blocks: perceptual classes and action elements. It is worth bearing in mind that however simple perception space might be, the underlying properties in sensor space could be very complex, and the work done by the perception system to generate perceptual classes correspondingly great.

Improving Accuracy

Even if an external assessor of the agent's learned behaviour has defined an ideal behaviour, the agent itself will not have done (otherwise, it would have nothing to learn) Therefore, it needs ways of identifying which regions of task space contain behavioural flaws. We do not detail exactly how an agent could achieve this; instead we discuss how it could hypothetically be facilitated by the four learning methods listed in Section 2.3.1:

Insight It may be that the prior learning episode has provided the agent with enough extra information for it to locate errors in its behaviour through mental processing, before the commencement of the next episode. These conjectures may need to be tested in the task environment itself, but can at least provide constraints on the improvement search.

Instruction The availability of this resource totally depends on the teaching programme in which the agent is involved. For example, it may be that instruction is provided incrementally; once a basic aptitude at the task is gained, a new, more detailed set of instructions is given to increase fine-level accuracy.

Observation If, after the end of an episode, observation of the task is still possible, then the behaviour learned up to that point can be used to predict the observed behaviour. The areas of task space which require most improvement are those where the predicted actions most often mismatch those of the demonstrator. This, of course, assumes that the demonstrator is competent at the task (i.e. is exhibiting better behaviour than has been already been learned by the agent), otherwise a reduction in accuracy could occur.

Trial-and-error How easy it is to detect errors using trial-and-error depends on how rewards and punishments are administered by the environment. If certain regions of task space often yield immediate punishment (or even consistently low reward), then it is these regions that are likely to require the most behaviour correction. On the other hand, if rewards are more long-term it becomes difficult (although not impossible) to identify in which states the most serious mistakes are being made.

It may be that once a fault is discovered, a change in mapping (i.e. assigning different actions to existing perceptual classes) or attention will correct it. However, it may equally be that the perceptual representation in the faulty region is just not rich enough to allow correct completion of the task. Provided that the classes in this region are not 'at full magnification' (i.e. being perceived at the finest possible grain), then sub-classes can be added under these classes to allow for more precise decision-making. For example, if a class represents an interval of an underlying continuum, that interval can be further subdivided to produce child classes; if a class represents a group of environmental features, then those features can be separately monitored, and so forth. To illustrate the latter case, suppose that an agent begins to to take into account the separate 'posture features' of another agent, where before it was just concerned with the agent's position. Through additionally monitoring head orientation, hand position, stance, etc., a more information-rich perceptual input space is available for decision-making.

When a more complex perceptual structure is required, it is possible to create a mapping which performs identically to the previous one. Suppose a 'parent' perceptual class p has been sub-divided into p_1 and p_2. Suppose also a skill s defines a mapping from a set of perceptual classes P which contains the parent class:

$$p \in P, s(P) = a, \text{ for some } a \in A \tag{2.2}$$

where a is an action element, and A is some subset of all such elements available to the agent. Then, generate the sets P_1 and P_2 from P by replacing p with p_1 and p_2 respectively. Now define:

$$s(P_1) = a, \text{ and } s(P_2) = a \tag{2.3}$$

and the behaviour is replicated with a new underlying representation. We can therefore state that, in theoretical terms, accuracy *increases monotonically* with perceptual complexity. In practise, a more exploratory choice of assignments is likely to be made, in which case performance may initially drop before rising to a higher level than was possible before.

Bearing these things in mind, it might seem that the best idea would be to perceive the whole of task space at the finest possible granularity, as that

would allow the best potential accuracy. There are two main problems with this; *reduction in efficiency* and *task overfitting*; which we describe briefly in turn.

An increase in the number of perceptual classes has a knock-on effect which must be distributed amongst other sub-systems:

- The attention system has to work harder to find a good selection.
- Behaviours must encode more complex associations (the trade-off between this and attention was touched upon in Section 2.3.2).
- Memory capacity must increase in either case.

In addition, searching the enlarged task space may require obtaining many more 'samples' via one of the four learning methods. Further discussion on efficiency can be found in the next section.

It could be that a task can be prescribed in sufficiently general terms for perfect behaviour to be learned across all tasks of that type. This is trivially true if, for example, the agent in question need only ever perform a single task (possibly repeatedly). However we should also consider the case of more difficult, dynamic situations and tasks in which any given behavioural specification will perform better on some tasks in a task class than others. Here, the more exactly a behaviour is tailored to a specific task (i.e. as it approaches 100% accuracy), the *worse* it is likely to perform in general (Kuris and Norton, 1985; Lawrence and Giles, 2000). For many tasks, general accuracy will increase with specific accuracy to begin with, and then at some point will start to decrease as the task is overfit. The aim should therefore be to find the point of maximal general accuracy, as opposed to seeking perfect behaviour in a specific task.

Improving Efficiency

Efficiency in this context refers to the amount of memory and search time required for learning. As discussed above, this is directly linked to the complexity of perception and action space; a reduction in complexity will lead to improved efficiency. To reduce perceptual complexity, we can invert the procedure outlined above. Groups of regions of task space which all point to the same action can be represented by a single feature (i.e. perceptual class), and redundant divisions can be removed by *perceptual merging*.

As task familiarity rises, it may be possible for the agent to build compound actions from elements already in its repertoire. For example, suppose an agent has learned to move to a visible object using the following map:

`object_left`	$\Longrightarrow$	`turn_left()`
`object_right`	$\Longrightarrow$	`turn_right()`
`object_ahead`	$\Longrightarrow$	`move_forward()`

If the agent can then create a compound action (in this case with a deictic goal) such as `move_to(object)`, the mapping can be re-expressed as:

`object_left`	$\Longrightarrow$	`move_to(object)`
`object_right`	$\Longrightarrow$	`move_to(object)`
`object_ahead`	$\Longrightarrow$	`move_to(object)`

Note now, however, that the perceptual divisions are unnecessary (at least as far as this part of the task space is concerned), and so can be removed via perceptual merging to give:

`object_visible`	$\Longrightarrow$	`move_to(object)`

This example illustrates the interdependence of perception and action representation. The expressed behaviour is identical to the original, but the new compound action has 'absorbed' some of the complexity from both spaces (Bryson, 2001, Section 6.5, discusses this deictic state / action trade-off in depth). The extent to which actions can be compounded in this manner depends greatly on the corresponding reasoning abilities of the agent and the dynamism of the task domain. It is unlikely that arbitrary reductions in complexity could be achieved using this method. Therefore, if improved efficiency is desired and action space cannot be further simplified, then perceptual merging / pruning is the only systematic way this can be achieved. At some point, as this process continues, *accuracy degradation* (which could be thought of as *task underfitting*) will occur. This is the point at which the perceptual representation ceases to be rich enough to adequately describe the task, resulting in a greater and greater number of mismappings being present in the learned behaviour. So, we have a clear trade-off between accuracy and efficiency within the constraints imposed by the agent's capabilities. An agent with infinite resources could simply aim to maximise accuracy across

a task class as outlined in the section above. A resource-bounded agent on the other hand must find the point at which accuracy is maximal given that real-time learning and decision-making must remain tractable.

2.3.4 Summary

In this chapter, we have formulated an agent-independent description of task learning which rests on three fundamental concepts. *Perceptual classes* can represent any region of sensor space, from raw low-level data to complex high-level concepts. *Action elements* can represent any executable action, from low-level motor commands to high-level co-ordinated sequences of movement. *Skills* or *behaviours* map perceptual classes to action elements, and too can be implemented at any level. Possessing innate biases and past experiences to constrain and guide learning, agents can make use of a combination of *insight*, *trial-and-error*, *observation* and *instruction* to hone their skills, depending upon their capabilities and circumstances. For best long-term results, this is likely to occur over a number of *episodes*, with the learner aiming to gradually improve both task accuracy and efficiency.

In the next chapter we set out in detail a task learning framework based upon this formulation, with a view to enabling others to implement it.

Chapter 3

General Task Learning Framework

Let us now move from the theoretical into the practical: given an agent situated within a task environment, how can the principles set out above be implemented? In Section 2.3.3, we proposed a cyclic approach to learning, and highlighted some of its advantages. We now sub-divide each learning cycle into the following four stages (see also Figure 3-1):

Stage 1: Learning Episode The agent uses exploratory behaviour to acquire knowledge from the task space, making use of any learning methods which are available (*insight*, *trial-and-error*, *observation* or *instruction*). Acquired behavioural data relating to *perception-action association*, *attention*, and *perceptual categorisation* (see Section 2.3) are stored in the episodic buffer.

Stage 2: Consolidation The association data in the episodic buffer are combined with those in long-term skill memory to create a new combined behaviour.

Stage 3: Testing The agent switches to the new behaviour and applies it to the task. Its relative performance is assessed using error metrics, which could be possessed both by the agent itself and by external observers. If performance is deemed to have improved, then the new behaviour is stored in place of the old one in long-term memory.

Stage 4: Reconfiguration The error data gathered during testing, along with

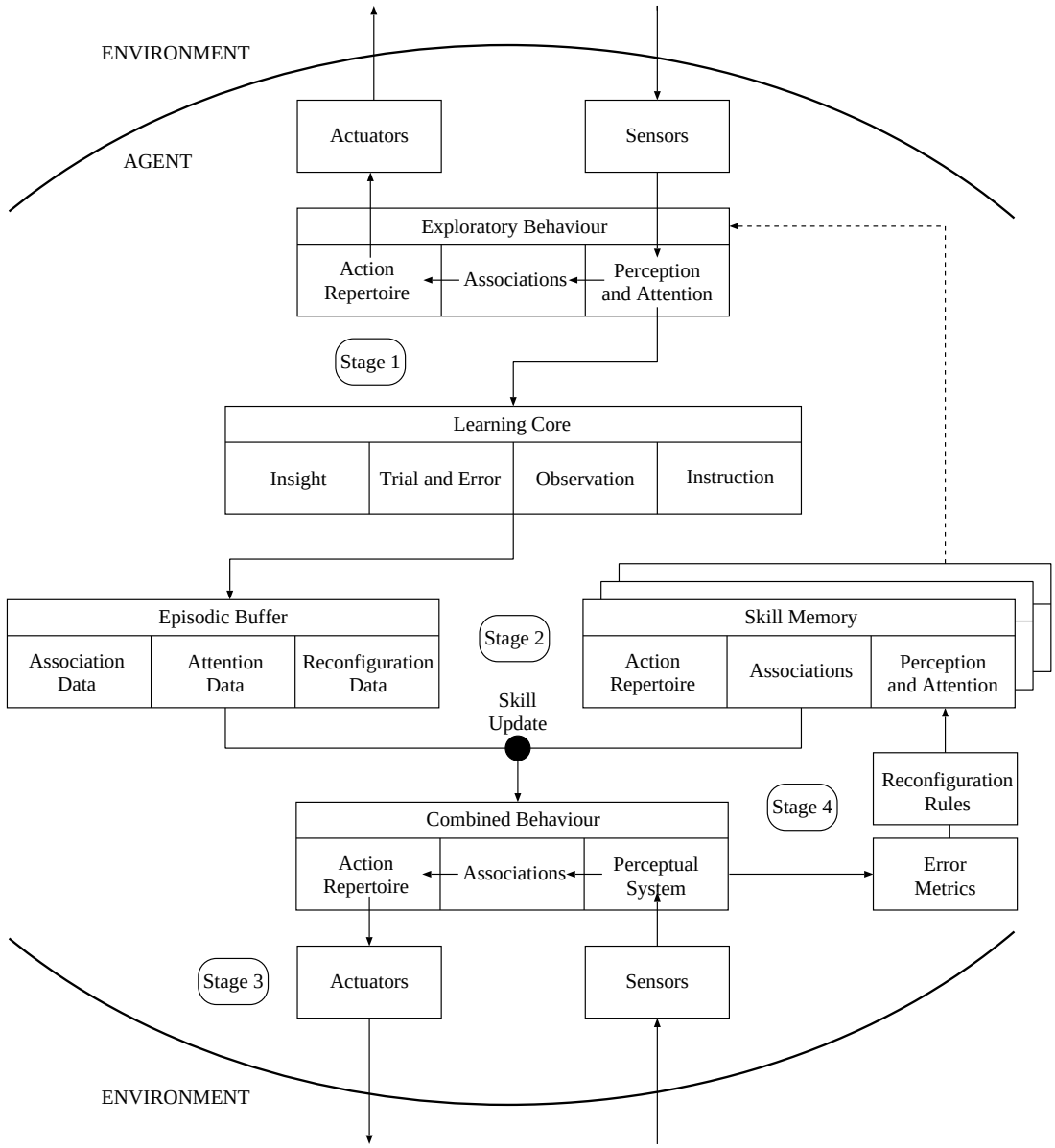

Figure 3-1: A schematic showing information flow in the General Task Learning Framework. The agent-environment interface is shown explicitly, and the approximate locations of each processing stage are also marked: Stage 1 – the agent gains task knowledge by exploring the environment; Stage 2 – newly acquired and prior task knowledge is combined; Stage 3 – the new combined behaviour is tested; Stage 4 – the agent's task representation is reconfigured if necessary.

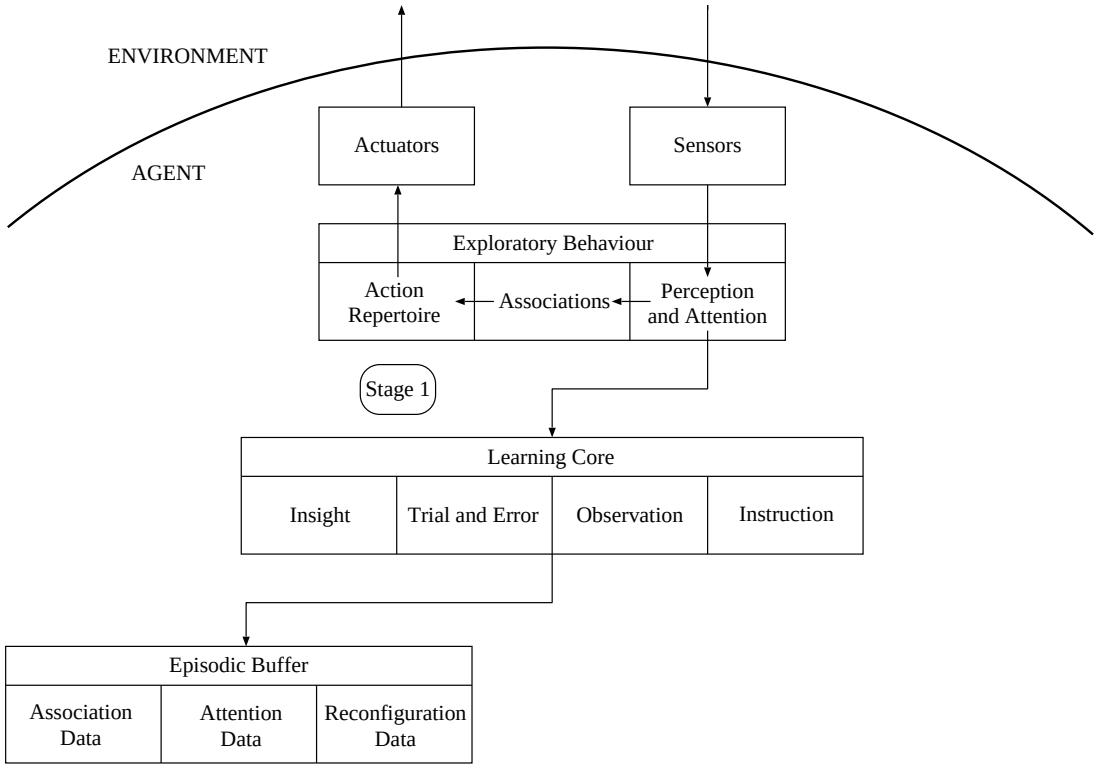

Figure 3-2: Stage 1: Learning Episode — the agent accumulates task knowledge via available learning methods during exploration of the environment.

the remaining data in the episodic buffer, are used to identify those parts of task space which are poorly attended or represented. Any applicable known rules are used to update the selection strategy or improve the representations accordingly.

Each of these stages deserves a detailed explanation, which follows:

3.1 Stage 1: Learning Episode

The purpose of a learning episode is to acquire task-related knowledge and store it in a buffer. This is done through processing raw sensor data, and using it to both guide exploratory behaviour and provide input to the agent's learning core (see Figure 3-2). We look at these in turn:

3.1.1 Processing Sensor Data

Task learning and execution requires response to changing environmental stimuli, and this is only possible if the agent possesses an array of sensors. As a physical property of the agent, they constitute part of the agent's innate learning biases (see Section 2.2.1). Few assumptions are made about the properties of these sensors, except that their input can be expressed as a series of attribute-value pairs. The attributes are symbols describing some varying sensed property and the values contain the current state of that property, and could also be a symbol, or equally a real, integer or boolean. A soccer-learning agent, for example, might receive the following raw sensor data: **{energy_level, *high*}**, **{goals_scored, *2*}**, **{see_ball, *true*}** and **{distance_to_goal, *18.2634*}**, which are all valid pairs. For the purposes of the framework, proprioceptive (e.g. **energy_level**, **goals_scored**) and exteroceptive (e.g. **see_ball**, **distance_to_goal**) sensors are treated identically. Knowledge such as **goals_scored** can be thought of as the input from a memory sensor.

We assume each sensor attribute can have at most one value at a given time, and thus the entire **sensor state** can be expressed as an ordered n-tuple, where n is the number of attributes, and each position corresponds to a different attribute: e.g. (***high, 2, true, 18.2634***). If a sensor attribute only receives an intermittent value (e.g. it can be deactivated), then it should still have a dedicated position in the tuple, which should contain a *null* value whenever another is not received.

The agent's perceptual system acts as a function, p, mapping sensor states onto perceptual states (that is, the set of all perceptual classes which currently apply):

$$p : \sigma \rightarrow \mathcal{P}(P) \tag{3.1}$$

where σ is the set of all possible sensor states, P is the set of all perceptual classes recognisable by the agent, and $\mathcal{P}(P)$ is the power set of P. A simple yet powerful example would be to define each perceptual class in terms of conditionals on some or all of the tuple quantities. If σ_1 represents the sensor state given above, and σ_1^4 represents the fourth tuple quantity (**distance_to_goal**), then we can define

a perceptual class `goal_near` by saying:

$$\sigma_1^4 < d \implies \texttt{goal\_near} \in p(\sigma_1) \tag{3.2}$$

That is, if **distance_to_goal** $< d$, then `goal_near` should be included in the perceptual state. It is equally possible to use a highly complex p function, to map very low-level raw sensor data (e.g. pixel Red-Green-Blue values from a camera) to high-level perceptual classes (e.g. `goalkeeper_ahead`). Once the perceptual system has output a perceptual state, the agent's attention strategy, a, then selects a subset of these perceptual classes for subsequent processing:

$$a : \mathcal{P}(P) \rightarrow \mathcal{P}(P),\ a(X) \subset X\ \forall\ X \in \mathcal{P}(P) \tag{3.3}$$

The most trivial possibility is that the entire state is retained, so $a(X) = X$. Another is that only the highest priority perceptual class is retained; this is the premise for our model of attention outlined in Section A.3. The perceptual classes output by the attention strategy serve two purposes: they provide input to drive the agent's exploratory behaviour, and they provide input to the learning core.

3.1.2 Exploratory Behaviour

Just as sensors are required for detecting environmental change, actuators are required to navigate the agent through perceptual states which might best facilitate learning, as well as for actually completing tasks. As explained in Section 2.2, an agent's action repertoire; that is, the set of action elements available for execution; is determined partly by the physical capabilities of the agent, and partly through experience. Action elements include both external and internal (i.e. cognitive) actions / skills, as both can cause changes to the agent's perceptual state. Also, they may or may not take (deictic[1]) control parameters:

- `move_forward_10()` – low-level action, externally executed, no parameters
- `move_forward(10)` – low-level action, externally executed, numerical parameter

[1]In this context, *deictic* control parameters make reference to variable elements of the task environment.

- `retrieve_ball()` – skill, externally executed, no parameters
- `assess_player_ability(goalkeeper)` – skill, internally executed, deictic parameter

From this last example, we see input and control being internally passed down to a lower-level skill, presumably from some high-level arbitration behaviour (see Section 3.5.1). This kind of internal interaction between modules is reminiscent of Minksy's *Society of Mind* model (Minsky, 1986).

It is assumed that the agent already possesses some kind of exploratory behaviour, which needs to be executed during this learning stage. If we recall from Equation 2.1 that a skill s maps perceptual states onto action elements ($s : \mathcal{P}(P) \rightarrow A$), then we can see how a composition of the systems / functions described above provides a map from sensor states onto action elements:

$$p \circ a \circ s : \sigma \rightarrow A \tag{3.4}$$

To execute a behaviour (in this case an exploratory behaviour), apply the agent's current sensor state to the composite function above, and execute the output action element.

For learning by observation, an exploratory behaviour might not require any action, but could include, say, moving to a good vantage point to view a demonstration. Learning by trial-and-error would require interaction with elements of the task environment, and so on. In some cases it may make sense for an agent to have two different attention strategies – a_E which outputs classes to the exploratory behaviour, and a_L which outputs classes to the learning core. To illustrate this, consider the act of following an expert soccer-playing agent to observe its shooting technique. This would require monitoring relationships such as distance and orientation with respect to the expert. Learning to shoot, on the other hand, requires monitoring distance from the ball, position of the goalkeeper, distance to the goal, etc. – the perceptual classes of interest in each case are very different.

3.1.3 The Learning Core

As the agent executes its exploration behaviour, the attention strategy periodically[2] outputs a subset of the perceptual state ($p \circ a_L : \sigma \rightarrow \mathcal{P}(P)$). We refer to this series of observations as the episode *trace*, $T = \{T_1, T_1, \ldots, T_N\}$, which is output in sequence to the learning core. The learning core is implemented as four separate modules; one for each of the four learning methods described in Section 2.3.1 – *insight*, *trial-and-error*, *observation* and *instruction*. The situation in which some of these modules are missing is simplified by the fact that each is assumed to operate independently and in parallel. The output from a_L is input to each available module, which then carries out its respective processing and outputs data to the episodic buffer. Before considering how each of these modules might generate data, we first look at the three categories into which they fall:

1. **Association Data** – if the module has learned something about the perceptual context in which certain actions should / should not be executed in order to complete the task (see Section 2.3.1).

2. **Perceptual Selection Data** – if the module has learned about which perceptual classes are more / less important with respect to the task (see Section 2.3.2).

3. **Perceptual Categorisation Data** – if the module has learned about which perceptual classes should be examined in more / less detail (see Section 2.3.3).

Association data are represented as triples in the set $\mathcal{P}(P) \times A \times \mathbb{R}$, where $\mathcal{P}(P)$ is defined as above, A is the set of all action elements in the agent's repertoire, and $\mathbb{R}$ is the set of real numbers. In other words, each triple represents a set of perceptual classes associated with an action element and a real number. The meaning of the real number is entirely dependent upon how skills are stored and updated in a given implementation (see Section 3.2.1). Suffice it for now to say that it could represent a frequency (in which case it would in fact be $\in \mathbb{N}$), weight, probability, utility value, strength, or indeed anything which might indicate a 'degree of associativity'.

[2]How this is regulated depends upon the specific agent implementation.

Perceptual selection data are represented as pairs in the set $\mathcal{P}(P) \times \mathbb{R}$, where definitions are as above. This time, the meaning of $\mathbb{R}$ depends upon how the attention strategy is stored (see Section 3.2.2), but will in some way represent 'degree of priority'.

Finally, perceptual categorisation data are also represented as pairs in the set $\mathcal{P}(P) \times \mathbb{R}$, where definitions are as above. They are used later in the learning cycle to reconfigure the perceptual system, as explained in Section 3.4. The meaning of $\mathbb{R}$ will depend upon exactly *how* they are used in a given implementation, but will in some way represent 'degree of scrutiny'.

So, how might such data be obtained? Part of the purpose of the framework is to allow researchers to answer this question by experimenting with different learning algorithms. Here, we look at some of the fundamental properties and requirements of each module, together with some examples to clarify and provide a baseline for further investigation.

The Insight Learning Module

To summarise the description given in the previous chapter, insight learning refers to the application of prior knowledge to form new hypotheses about previously unseen tasks. The primary example of insight as far as this framework is concerned, is the transferral of knowledge from a learned skill to a new one. Therefore, the primary role of the insight learning module is to search skill memory for knowledge that may be relevant to a new task, and present it in a form that can be integrated with the skill being learned. In turn, exploratory behaviour for this purpose amounts to surveying the task environment for stimuli which might match existing skills. Any relevant knowledge possessed by the agent which is *not* a component of an existing skill, can instead be represented by rules stored in the module itself (see below).

For example, suppose our soccer-learning agent has no knowledge of the game whatsoever, but instead is a skilled rugby[3] player. When the agent is placed on a soccer pitch, it may immediately recognise its context due to its familiarity with rugby pitches; e.g. the perceptual class `on_pitch` is included in the set T_n passed to the learning core. However, it may initially fail to notice the soccer

[3]Similar to the game of American football, except that play can continue if the ball touches the ground.

ball on the pitch; $\texttt{see\_ball} \notin T_n$. By searching skill memory, suppose that the insight learning module finds that rugby-playing might be a relevant skill, since its associated attention strategy regards `on_pitch` as a salient class. The same attention strategy may also regard `see_ball`, `ball_far` and `ball_near` as salient classes, in which case the insight learning module could output the perceptual selection data

$$(\{\texttt{see\_ball}\}, 1)_s, (\{\texttt{ball\_far}\}, 1)_s, (\{\texttt{ball\_near}\}, 1)_s$$

suggesting that the attention strategy associated with the new soccer-playing skill also selects those perceptual classes. If in the next learning episode:

$$\texttt{see\_ball}, \texttt{ball\_far} \in T_n$$

this indicates that the attention strategy has indeed been changed. Note that this agent's ability to recognise the soccer ball as a type of ball even though it differs in size, shape and colour from a rugby ball, is represented by the ability of its perceptual system to appropriately map the different regions of sensor space to the same perceptual classes.

Given T_n as above, suppose that the insight learning module finds that the rugby skill function, s_r, defines the following association:

$$s_r(\{\texttt{see\_ball}, \texttt{ball\_far}\}) = \texttt{move\_to\_ball()}$$

and therefore outputs the association datum

$$(\{\texttt{see\_ball}, \texttt{ball\_far}\}, \texttt{move\_to\_ball()}, 1)_a$$

suggesting that the same association should be made by the new skill. This insight happens to be correct. If next time, however:

$$\texttt{see\_ball}, \texttt{ball\_near} \in T_n$$

this could lead the insight learning module to output

$$(\{\texttt{see\_ball}, \texttt{ball\_near}\}, \texttt{pick\_up\_ball()}, 1)_a$$

an incorrect insight, as this is against the rules of soccer[4]. This demonstrates the importance of testing skills learned by insight alone.

[4]Except for a goalkeeper standing within the goal area.

If we assume that the soccer-learning agent has no rugby-playing skill, but has still learned by some other means that when on a sports field the ball is important, this could be represented within the insight learning module by the following rule:

$$\text{IF } \texttt{on\_pitch} \in T_n, \text{ THEN output } (\{\texttt{see\_ball}\},\ 1)_s$$

Such rules are searched and matched in exactly the same way as skill memory.

The fact that insight learning relies chiefly on internal search has unique benefits. Firstly, its exploratory behaviour requirements are relatively low, and are likely to be more than satisfied by allowing the more 'demanding' modules to control exploration. This means that insight can run 'in the background', acting almost like an advisor; periodically making suggestions which inform the learning process for the other modules. Secondly, an agent may still be able to make use of insight even when not occupying the task environment. Recall that an agent's perceptual state can include memories of previously observed perceptual states or classes. These remembered stimuli can be used for learning by insight in the same way as 'live' stimuli, allowing insight to also run 'offline'. In this case, insight functions comparably to Stein's concept of robotic 'imagination' (Stein, 1994).

The Trial-and-Error Learning Module

Task learning by trial-and-error involves interpreting feedback signals generated through interacting with elements of the task, and adjusting behaviour accordingly. Feedback signals can originate in the environment, in which case they arrive in GTLF via the perceptual state (as do all environmental stimuli), and must be identified and translated by the trial-and-error learning module. They could also be generated internally by *reward functions*, which comprise part of the module itself. Exploratory behaviour for the purpose of learning by trial-and-error should seek to attend to any relevant reward signals present in the environment, and carry out task interactions which yield a maximal amount of new information for processing by the learning module, as opposed to necessarily attempting to complete the task optimally. This technique of learning a deterministic behaviour for completing a task while executing a different (but probably related) exploratory behaviour is known as *off-policy learning* in the

Reinforcement Learning literature (Sutton and Barto, 1998). The later Testing stage (Section 3.3) involves receiving feedback while executing the current best estimate of task behaviour; *on-policy learning*. In some task scenarios there may be a trade-off between information gain and the cost of that gain to the agent (in terms of resources, damage, continued opportunity to complete the task, etc.), which should be considered when choosing an exploratory behaviour.

To illustrate these features of learning by trial-and-error, we move from our sporting case study to the more safety-critical task of minefield clearance. Suppose that the mines, when deactivated, emit an audible shutdown tone. This tone constitutes an external feedback signal which should be interpreted by the trial-and-error learning module as a reward. In addition, suppose the module contains a reward function which outputs a penalty signal in proportion to the distance travelled by the agent, reflecting its prior knowledge that roaming around a minefield is best avoided. To gain maximal knowledge about the mines, it would in theory be advantageous to experiment with all possible manipulations, in order to determine which would best contribute to an optimal completion of the task. However, since certain operations could result in a very significant penalty (i.e. destruction of the agent), exploratory behaviour should be appropriately restricted.

Of all the learning methods, trial-and-error has the most widely studied and accepted formalism with respect to implementation in autonomous agents: Reinforcement Learning (RL). The use of this formalism is nowhere assumed in GTLF; the researcher can use any methodology that is compatible with the framework. However, by using RL as our trial-and-error learning example both in this chapter and experimentally (see Section 7.1.2), we hope to demonstrate its compatibility with GTLF, as well as clarifying the role of this learning module for the widest possible interested audience. In Section 3.2.1, we look at how RL action-value matrices can be used to define skill functions, and at a commonly-used method for updating these matrices. For now, we look at how the general formulation of the RL problem, as shown in Figure 3-3, relates to the role of the trial-and-error learning module. To summarise:

1. The environment sends a state signal, s_t to the agent.

2. The agent executes an action, a_t, determined by the policy it is following.

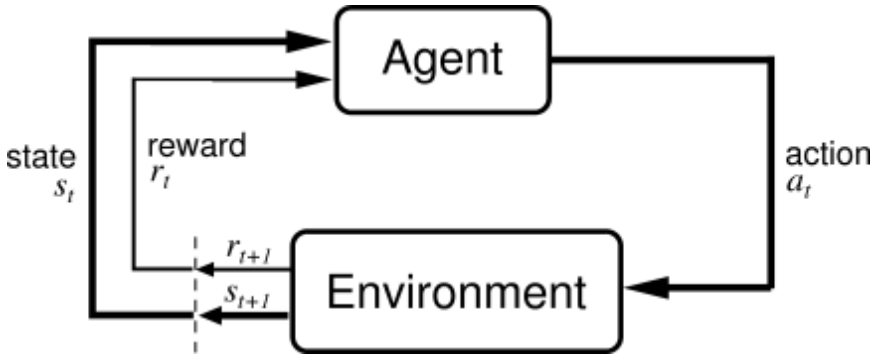

Figure 3-3: The Reinforcement Learning paradigm: having entered state s_t at time step t, the agent executes action a_t, determined by the policy it is following. The environment then responds with reward signal r_{t+1}, the agent is moved to state s_{t+1}, and the process iterates. Diagram by Sutton and Barto (1998).

3. The environment sends a reward signal, r_{t+1}, determined by the new state.

4. The environment sends a new state signal, s_{t+1}.

States in RL are assumed to be discrete, and can therefore be associated with the discrete perceptual states used in GTLF. Similarly, RL actions can be associated with our discrete action elements. The reward signal, r_{t+1}, is a real number, and in GTLF it is derived from a combination of external stimuli and internal reward functions as described above. If we assume that the action element being executed by the agent at any given time is represented in the agent's perceptual state (i.e. that the agent is aware of its own actions), then the perceptual state signal received by the trial-and-error learning module during exploration contains all the information necessary to create the $\{(s_t, a_t, r_{t+1})\}$ sequence required for skill update.

For example, suppose our minefield-clearing agent detects an armed mine, moves toward it, disarms it, and then continues searching. The perceptual state sequence could be:

$$\begin{aligned} \mathrm{P}_1 &= \{\texttt{armed\_mine\_detected}, \texttt{moving\_to\_mine}\} \\ \mathrm{P}_2 &= \{\texttt{armed\_mine\_near}, \texttt{disarming\_mine}\} \\ \mathrm{P}_3 &= \{\texttt{hear\_shutdown\_signal}, \texttt{searching\_for\_armed\_mine}\} \end{aligned}$$

resulting in the following output from the trial-and-error learning module:

$$(\{\texttt{armed\_mine\_detected}\}, \texttt{move\_to\_mine()}, -0.1)_a$$

$$(\{\texttt{armed\_mine\_near}\},\ \texttt{disarm\_mine()}, 1)_a$$

where the reward of -0.1 is generated by an internal reward function which penalises movement (see above), and the reward of 1 results from the module's translation of `hear_shutdown_signal` into a reward[5]. Even though moving toward an armed mine is necessary for success in the task, the immediate reward received is negative (since movement in general is bad). This is an example of the *credit assignment problem*, and its effects depend upon which algorithms are used to update the policy (see Section 3.2.1).

The basic formulation of the RL problem that we have considered so far, assumes that tasks can be described as discrete time Markov Decision Processes (Russell and Norvig, 2003, ch. 17). In our implementation of GTLF, we show how this can be extended to continuous time Semi-Markov Decision Processes (see Section 7.1.1).

The Observation Learning Module

Observing other agents interacting with elements of a task can provide information to speed the learning of that task. This can be extended to task-related events caused by forces not generated by agents (e.g. gravity, wind, etc.), but here we concentrate on the former case of *social* learning. It should be noted that, compared with the highly complex and well-founded categorisations proposed in the ethology literature (Whiten and Ham, 1992; Zentall, 2001), we make a simple, functional distinction between *imitation* and other nonimitative types of social learning. Specifically, learning is by imitation if the agent being observed is carrying out the task being learned.

Regardless of their intentions, we refer to any agent deemed a source of task knowledge by the learner as an *expert*. We refer to the behaviour about which the expert is relaying (noisy and possibly incorrect) information to the observer,

[5]Note here that we have assumed that the agent is capable of mapping its perception of its actions (e.g. `moving_to_mine`) to executable action elements (e.g. `move_to_mine()`). This is a safe assumption, since we have already assumed the *inverse* of this map exists: the agent was able to include the actions it was executing in the perceptual state. It is also worth noting that this map is equivalent to the *action correspondence library* required for imitation learning (see below), except that it relates the agent's *own* observed actions to its action repertoire, rather than the actions of another agent. In biology, such a map could be seen as a *forward model* of the kind learned by infants through *body babbling* (Meltzoff and Moore, 1983).

and which the observer should attempt to recreate, as the *target behaviour*. It need not be optimal or 'correct' in any absolute sense; in fact, it can be defined arbitrarily. The problem of deciding which agents will demonstrate *good* target behaviours, and therefore should be treated as experts, is a complex one in its own right (Schlag, 1998), and beyond the scope of this module.

Consider a common scenario in which a learner is attempting to acquire a skill by observing an expert. Exploratory behaviour for this purpose should obviously seek to put the learner in a position where it can observe the task being carried out. The exact nature of this behaviour depends upon the task, but some work suggests that following as opposed to statically observing a task demonstrator can lead to improved learning performance due to closer perceptual matching (Hayes and Demiris, 1994; Billard, 2002). As for any external source, salient information generated by the expert's actions must be contained within the perceptual state. However, there remains the question of how perceptual classes relating to the state and actions of the expert can be translated by the learner into executable task behaviour. This is the role of the *correspondence library*.

We refer to those perceptual classes which describe the expert's state and actions as *allocentric*. These must be translated into *egocentric* perceptual classes and action elements; that is, describing the state and actions of the learner; in order to be usable for subsequent skill update. To this end, the correspondence library contains two types of correspondences. Perceptual correspondences associate allocentric perceptual classes relating to state with egocentric perceptual classes relating to state. For example, the allocentric class `expert_sees_ball` could be associated with the egocentric class `see_ball`. Action correspondences associate allocentric perceptual classes relating to action with (egocentric) action elements. For example, `expert_running_to_ball` could be associated with the action element `run_to(ball)`. In the examples above, the implication is that the correspondences are between conspecific agents, but this need not be the case. It is equally possible for the library to contain correspondences for agents with dissimilar embodiments (Alissandrakis, 2003; Alissandrakis et al., 2002, 2005, 2007). For example, if a robot on tracks was imitating a human, the action correspondence above might become `expert_running_to_ball` $\Rightarrow$ `roll_to(ball)`.

Given that the learner can relate the expert's behaviour to its own, what cues should prompt the observation learning module to output data to the episodic

buffer? We give an example in our implementation (see Section 7.1.2), but for now we highlight what we consider to be some desirable general principles:

1. Perceptual classes which always give rise[6] to action should increase in attentional priority.

2. If a given perceptual state reliably gives rise to the *same* action, an association between the state and action should be created (or strengthened).

3. If a given perceptual state gives rise to many different actions, this is an indication that the task is not well represented in this region of task space. Either the perception / action configuration is inappropriate, or the correspondence library contains incorrect associations (or both).

Each of these principles constitutes a condition which should be evident from the (sequence of) input perceptual states. The first should output a perceptual selection datum; the second an association datum; and the third a perceptual categorisation datum.

The Instruction Learning Module

For the purposes of GTLF, we define learning by instruction as the acquisition of task knowledge via explicit communication with another agent. Implicit instruction, through deliberate demonstration for example, falls under the remit of the observation learning module. However, communication could be indirect, via written instructions for example.

Instructions, like observations, must be received via the perceptual state. For example, suppose an agent can discern a set of perceptual classes of the form `heard_word_`W_i, where each W_i corresponds to a different spoken word. Consider the sequence of words (sentence) $W_1, W_2, \ldots, W_n$, given as instruction to the learner by a teacher agent. Ignoring the other perceptual classes, and providing that the learner is attending to the teacher, this would correspond to the perceptual state sequence:

[6] If an action is initiated while the agent occupies a given perceptual class, we say that the perceptual class *gives rise* to the action.

$$\begin{aligned} \mathrm{P}_1 &= \{\texttt{heard\_word\_}W_1\} \\ \mathrm{P}_2 &= \{\texttt{heard\_word\_}W_2\} \\ &\vdots \\ \mathrm{P}_n &= \{\texttt{heard\_word\_}W_n\} \end{aligned}$$

The instruction learning module must attempt to *interpret* this state sequence; that is, output appropriate data to the episodic buffer.

The simplest case involves instructions issued as commands throughout a task learning episode. Such commands may consist of just one or two words, and can thus be interpreted by referring to a relatively basic lexicon. State information need not necessarily be part of the instruction, since this can be inferred from the state of the learner at the time the instruction was issued. For example, suppose our soccer-learning agent experiences the perceptual state:

$$P_1 = \{\texttt{have\_ball}, \texttt{heard\_word\_shoot}\}$$

The interpreter looks up `heard_word_shoot` in the agent's lexicon, and finds it is associated to the action element `shoot()`. The interpreter can then associate this action element to the remainder of the learner's perceptual state at the time of the command, to generate the association datum:

$$(\{\texttt{have\_ball}\},\ \texttt{shoot()}, 1)_a$$

Similarly, commands such as "Watch!" could be used to generate perceptual selection data, since they imply that the perceptual classes currently contained within the perceptual state should be given an increased priority. Commands such as "Bad!" could be used to generate perceptual categorisation data, as they may indicate that the classes contained in the perceptual state need to be reconfigured in order to allow correct behaviour[7]. This technique has been used successfully in tandem with learning by observation to teach robots to traverse a maze (Nicolescu and Matarić, 2007).

As far as more complex interpretation algorithms are concerned, the module can in principle implement any that are capable of parsing a sequence of symbols. This includes advanced Natural Language Processing (NLP) methods, which may reference complex grammars (Manning and Schütze, 1999). Rather

[7]Such commands could also indicate an incorrect perception-action association.

than merely barking orders, such systems could allow teachers to issue (sets of) instructions in advance (e.g. "If you have the ball, shoot."). If an instruction cannot be exactly interpreted, then the learner can either ignore it, or attempt to infer an approximate interpretation. For example, a learner that is incapable of parsing the complete sentence given above may still be able to infer its meaning by recognising the keywords "have", "ball" and "shoot". Of course, such approximate inference introduces the possibility of making incorrect interpretations and consequently incorrect skill updates.

All of the above discussion applies to any sequential form of instruction, e.g. `read_word_`W_i for written words, `observed_gesture_`G_i for symbolic gestural communication, and so on. However, instructions could also be given in parallel, e.g. the spoken command "Pass!" accompanied by a referential gesture indicating another player.

Clearly, opportunities for learning by instruction are largely dependent upon the actions of other agents. The learner's exploratory behaviour can only really seek to take full advantage of those opportunities. Firstly, if a teacher is available but not present, then it should be sought out; in other words, the learner should get into a position where instructional classes may appear in the perceptual state. Secondly, if a teacher is teaching, pay attention; the learner's attention strategy should select the instructional perceptual classes. As with learning by insight, the minimal requirements of this exploratory behaviour (given the availability of a teacher) renders learning by instruction ideal for combination with other methods. Instructions could be used to guide the trial-and-error learning process, or augment the observation learning process, for example.

Summary

Data continue to accumulate in the buffer until either the learning episode ends or the buffer becomes full. In the latter case, any subsequent data are either discarded or overwrite the prior contents, whichever is preferable. The size of the buffer is a physical constraint of the individual agent, and affects (amongst other things) the frequency with which it must transfer between learning and consolidation (see below).

Endel Tulving, who first proposed the concept of episodic memory in humans (Tulving, 1972), describes it as 'memory for personally experienced events'

or 'remembering what happened where and when' (Tulving, 2001). Baddeley (2000, 2001) proposes the addition of an episodic buffer to his model of working memory which acts as a 'limited-capacity temporary storage system that is capable of integrating information from a variety of sources'. GTLF's episodic buffer fits both Tulving's description, since it records a sequence of experienced (grounded) perceptual classes, and Baddeley's model, since it is a finite buffer which integrates data from multiple learning modules. The idea has gained further credence recently, with Nuxoll and Laird (2007) adding episodic learning capabilities to the SOAR cognitive architecture.

Before we look at how the buffer contents are used for skill update, we further consider the case in which multiple learning methods are concurrently available. For example, it is conceivable that an agent with reasoning abilities (*insight*) could be given a demonstration of a task (*observation*) accompanied by verbal prompts (*instruction*) while being allowed to practise it (*trial-and-error*). If the agent is capable of attending to multiple information sources, then this is clearly the best option so that no information is 'wasted'. Otherwise the exploratory behaviour, along with its attention strategy, will have to arbitrate between sources. It may be wise to give priority to those which are dependent upon the actions of other agents when they are available (e.g. *imitation* and *instruction*). On the other hand, if the information quality from some sources is particularly poor, an exploratory behaviour which can determine this and switch attendance accordingly would be beneficial.

3.2 Stage 2: Consolidation

Upon completion of a learning episode, the agent should attempt to consolidate what it has just learned (see Figure 7-1). Whether this must be done 'online' (i.e. while the agent remains active) or can be done 'offline' (i.e. while sleeping, dormant, daydreaming, etc.) depends on the capabilities of the agent and, moreover, the circumstances of learning. For example, an agent that is being pursued through a maze by a predator does not have the luxury of offline consolidation available to a robot which is learning to stack blocks in a lab. This example also highlights another possibility: that consolidation could be deferred until a more convenient time.

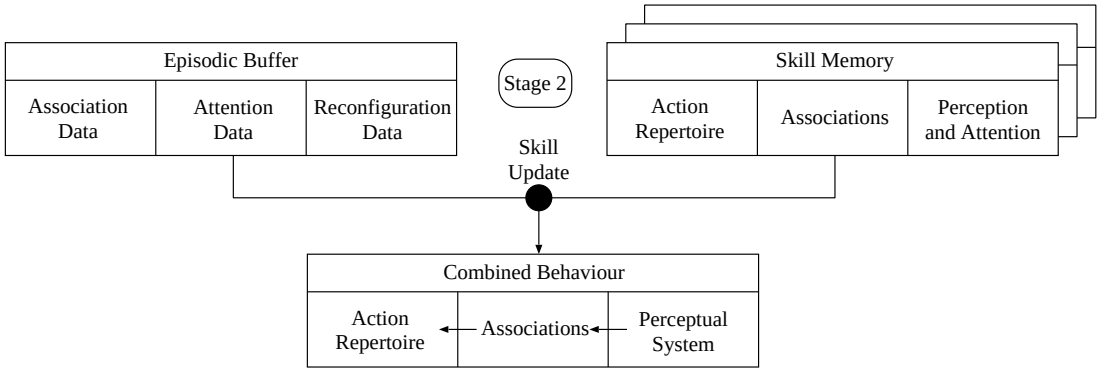

Figure 3-4: Stage 2: Consolidation — newly acquired and prior task knowledge is combined.

Practically speaking, the purpose of consolidation is to:

1. Create a hypothetically improved skill by using the newly acquired association data in the episodic buffer to adjust previously defined perception-action associations.

2. Similarly improve the agent's attention strategy by using the new perceptual selection data to adjust the previous strategy.

We now look at how each of these can be achieved.

3.2.1 Creating an Improved Skill

The first stage in creating an improved skill is to take the original skill stored in long-term memory, and copy it to working memory. This way, if the updates that are made result in a decline in task performance (see Section 3.3), the new skill can be discarded and the former one retained. As we explained above, association data takes the form (P_i, a_i, w_i), where P_i is a set of perceptual classes, a_i is an action element, and w_i is a real number. The meaning of w_i, and in fact the whole combination process, is governed by the way a particular agent stores skills in long-term memory. This is not specified in the framework, but we now look at some alternatives and explain possible combination approaches for each.

Tabular Representations

Recall that a skill is a function $s : \mathcal{P}(P) \rightarrow A$. One way of storing such a function would be to tabulate the members of $\mathcal{P}(P)$ against the members of A. At each cell position, a value indicating the agent's measure of confidence in that assignment (e.g. probability, weight, frequency count, etc.) can be stored. For example, suppose $P = \{p_1, p_2\}$, and $A = \{a_1, a_2\}$. Then the table would look as follows:

	a_1	a_2
$\emptyset$	$U(\emptyset, a_1)$	$U(\emptyset, a_2)$
$\{p_1\}$	$U(\{p_1\}, a_1)$	$U(\{p_1\}, a_2)$
$\{p_2\}$	$U(\{p_2\}, a_1)$	$U(\{p_2\}, a_2)$
$\{p_1, p_2\}$	$U(\{p_1, p_2\}, a_1)$	$U(\{p_1, p_2\}, a_2)$

and so U forms a matrix referenced by a subset of P and an action in A. The skill function can be defined by selecting the action element with the greatest value for each row, or the selection could be stochastic; in fact any process which makes a unique selection would define a valid function.

Consider the association datum (P_i, a_i, w_i), stored in the episodic buffer. Due to the complete enumeration of the task space, this datum will match exactly one cell of the matrix U. The value attached to the datum, w_i, and the value in the matched cell, $U(P_i, a_i)$, can then be merged using an appropriate *merging function*, m:

$$m : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \tag{3.5}$$

If the values represent frequency counts, then these can simply be added and the result stored in the skill under construction:

$$U_{new}(P_i, a_i) = m\big(U_{old}(P_i, a_i), w_i\big) = U_{old}(P_i, a_i) + w_i \tag{3.6}$$

If they represent weights, then some kind of weighted sum could be appropriate; a linear interpolation, for example:

$$U_{new}(P_i, a_i) = m\big(U_{old}(P_i, a_i), w_i\big) = (1 - \alpha)U_{old}(P_i, a_i) + \alpha w_i \tag{3.7}$$

where $0 \leq \alpha \leq 1$ represents the degree of influence new data have on the new behaviour, compared to the old behaviour. If α is fixed, then new data will always

have the same impact, no matter how well practised the skill in question is. For example, $\alpha = 1$ indicates that new data should just be copied directly to the new behaviour. Suppose instead that α is set to equal $\frac{1}{n}$, where n scales with the number or length of episodes so far spent learning the skill. Then each new episode has a lesser effect, as skill proficiency (theoretically) increases; this could be a desirable property.

Also, if weight updates have occurred on row i, then it may be necessary to renormalise that row:

$$U_{norm}(P_i, a_i) = \frac{U_{new}(P_i, a_i)}{\sum_j U_{new}(P_i, a_i)}, \forall\ i \tag{3.8}$$

The simple examples given above only make use of each datum individually. One of the advantages of episodic learning is that batch learning algorithms which reference multiple data can be used for skill update if desired. For example, suppose U represents an action-value matrix, as would be common in Reinforcement Learning problems. The update formula for *one-step Q-learning* (Watkins and Dayan, 1992) is:

$$U_{new}(s_t, a_t) = U_{old}(s_t, a_t) + \alpha\big[r_{t+1} + \gamma\ \max_b[U_{old}(s_{t+1}, b)] - U_{old}(s_t, a_t)\big] \tag{3.9}$$

where s_t is the state at time t, a_t is the action taken at time t, s_{t+1} is the new state reached, r_{t+1} is the reward received, α is the learning rate, and γ is the discount factor. Recall from Section 3.1.3 that the trial-and-error learning module outputs data such that $(P_i, a_i, w_i) = (s_t, a_t, r_{t+1})$. Substituting into Equation 3.9 we have:

$$U_{new}(P_i, a_i) = U_{old}(P_i, a_i) + \alpha\big[w_i + \gamma\ \max_b[U_{old}(P_{i+1}, b)] - U_{old}(P_i, a_i)\big] \tag{3.10}$$

which references two consecutive data in the episodic buffer: (P_i, a_i, w_i) and $(P_{i+1}, a_{i+1}, w_{i+1})$.

Alternative Representations

Tabular skill representation is able to express any possible assignment of action elements to perceptual classes, but such expressiveness comes at a price. The table described above contains $|A|.|\mathcal{P}(P)|$ cells. Using a well-known result from

set theory, we have:

$$|A|.|\mathcal{P}(P)| = |A|.2^{|P|} \tag{3.11}$$

In other words, although the size of the table grows linearly with the size of the agent's action repertoire, it grows exponentially with the number of perceptual classes it can discern. For example, an agent with 10 action elements that can recognise 10 different perceptual classes must assign confidence values to 10240 perception-action combinations for each skill[8]. Apart from space (i.e. memory) constraint considerations, this could pose a significant learning problem for complex task spaces, as the number of behavioural data which must be sampled by the learning modules also increases exponentially.

There are at least two solutions to this problem. The first is to reduce the number of classes passed to the learning core by the attention strategy, a_L (see Sections 2.3.2 and 3.1.3). This effectively requires the skill to map (i.e. find values for) only a subset of $\mathcal{P}(P)$, without having to remove classes from P altogether which might be salient. This method rests upon the assumption that at any one time, only a few perceptual classes will actually affect the agent's choice of action, and therefore, provided the strategy is good, little if any behavioural expressivity will be lost. This assumption must hold true in the case of human vision, for example, which only allows a few percepts to be concurrently attended to (Rensink, 2000). The most restrictive strategy possible would be for a_L to pass on only what it considers to be the highest priority perceptual class, which reduces the table size from $|A|.2^{|P|}$ to just $|A|.|P|$ (only a map from the singleton subsets must be defined).

The second solution involves *approximating* the skill function using a statistical classification algorithm. The classification problem in question is to find:

$$\mathrm{P}(a_n|P_n),\ \forall\ a_n \in A,\ P_n \in \mathcal{P}(P) \tag{3.12}$$

i.e. what is the probability that a_n should be chosen as an action element given that perceptual classes P_n apply? In this case, the skill is represented by the parameters of the classifier, and the output probabilities serve the same purpose as the confidence values described above. Here the reduction in learning / space complexity results from the fact that, as long as an appropriate classifier is chosen,

[8]That is, as long as the skill uses the same perceptual categorisation and action repertoire.

the number of parameters which need tuning should be significantly fewer than the number of values necessary for the tabular representation. To approximate $\mathrm{P}(a_n|P_n)$, a classifier needs *training samples* of the form (P_n, a_n) in order to adjust its parameters. The data in the episodic buffer is of the form (P_n, a_n, w_n), and can therefore be used for training, with w_n in this case representing a frequency count for each datum. Examples of using Decision Tree and Multi-Layer Perceptron classifiers for this purpose are given in Section 6.2. It should also be noted that these two solutions can be used in tandem, i.e. using the agent's attention strategy to restrict the types of training samples available to a classifier.

3.2.2 Creating an Improved Attention Strategy

As well as improving perception-action associations, the agent may also have learned enough to improve its attention strategy (see Section 2.3.2). Recall that an attention strategy selects a subset of the perceptual state to pass onto the learning core and / or the agent controller during exploration and task execution (see Section 3.1.1). If an agent has multiple attention strategies, we focus here on a_L; that is, the strategy that filters data relating to the task being learned to the learning core. Any strategy a_L must satisfy:

$$a_L : \mathcal{P}(P) \rightarrow \mathcal{P}(P),\ a(X) \subset X\ \forall\ X \in \mathcal{P}(P) \tag{3.13}$$

Now, a full enumeration of this function would require tabulating members of $\mathcal{P}(P)$ against their subsets in $\mathcal{P}(P)$. Since every member of $\mathcal{P}(P)$ has at least one subset (itself) and at most $2^{|P|}$ subsets (for the element $P \in \mathcal{P}(P)$), this implies there would be between $2^{|P|}$ and $2^{2.|P|}$ cells in such a table – in any case, exponential in the number of discernable perceptual classes. Since the single role of the attention system is to make the learning and skill execution processes more efficient, it would seem unnecessary (albeit perfectly legal) to consider such a complex representation. Instead, we briefly look at some simpler, more practical strategies.

The first, mentioned above, is the trivial strategy which passes on the entire perceptual state:

$$a(X) = X,\ \forall\ X \in \mathcal{P}(P) \tag{3.14}$$

This strategy forces the skill representation to deal with the full complexity of the agent's current perceptual configuration. In fact, many of the agents described in Part II are able to use this strategy, due to the simplicity of their categorisations. Perceptual selection data can have no effect here, and can be discarded.

Another possibility would be to impose a total order on P, and then return the n ($< |P|$) highest priority classes. For example, if $P = \{P_1, P_2, P_3\}$, the order is (P_2, P_1, P_3) and $n = 1$, we have for $X \in \mathcal{P}(P)$:

X	$\emptyset$	$\{P_1\}$	$\{P_2\}$	$\{P_3\}$	$\{P_1, P_2\}$	$\{P_1, P_3\}$	$\{P_2, P_3\}$	$\{P_1, P_2, P_3\}$
$a(X)$	$\emptyset$	$\{P_1\}$	$\{P_2\}$	$\{P_3\}$	$\{P_2\}$	$\{P_1\}$	$\{P_2\}$	$\{P_2\}$

Recall perceptual selection data is of the form (P_1, w_1), where $w_1 \in \mathbb{R}$. Here w_1 could represent absolute or relative ranking information; i.e. $w_1 = 1$ implies the paired perceptual class should move to position 1 in the order (absolute), or move up 1 position in the order (relative). In either case the displaced classes would move down.

A partial order could be represented by a set of pairs of the form $P \times \mathbb{N}$, e.g. $\{(P_2, 1), (P_1, 1), (P_3, 3)\}$; in this case P_2 and P_1 have equal highest priority. Perceptual selection data can be used to update a partial order in a similar manner to a total order. Yet another possibility is for the pairs to have the form $P \times \mathbb{R}$, where the real number represents, say, probability of saliency: $\{(P_2, 0.7), (P_1, 0.6), (P_3, 0.2)\}$. Perceptual classes could then be selected only if this probability exceeds some threshold, for example. Updating these probability values using perceptual selection data (e.g. $(P_1, 0.65)$) would require a merging function similar to that discussed in Section 3.2.1.

It should be noted that all of the above strategies assume that the saliency of a perceptual class is independent of the other classes. It is, of course, possible to create strategies in which, for example, P_1 only has high priority when P_2 is present. In fact, as we said at the start of this section, arbitrarily complex strategies are possible, although the problem of allowing them to be updated when new data arrive must also be solved.

3.3 Stage 3: Testing

At this stage, the agent has hypothesised a new combined skill, along with a new attention strategy (if one was required), but these need to be tested in

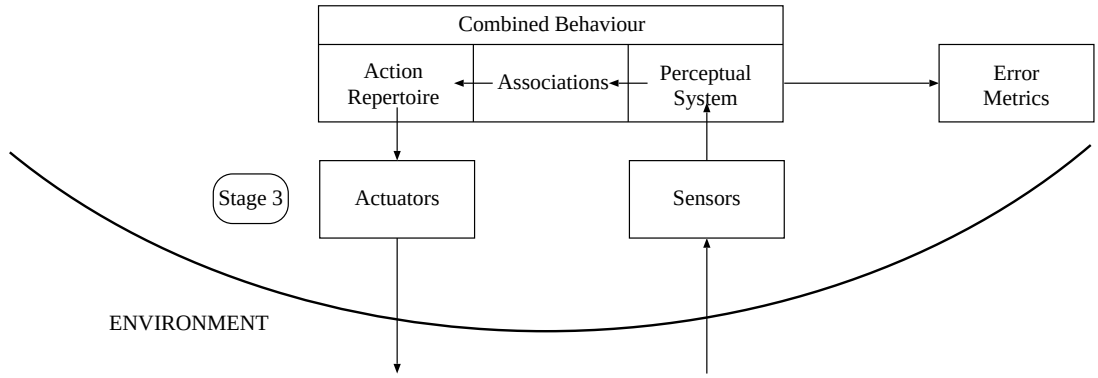

Figure 3-5: Stage 3: Testing — the new combined behaviour is executed, and the extent of improvement assessed using error metrics.

order to determine the extent of performance improvement (see Figure 3-5). It is of course possible that during consolidation, the agent's environment will have changed such that the task in question is no longer present, or that it is somehow not appropriate for the agent to attempt the task immediately. In these cases, the combined behaviour remains stored until the task becomes both available and executable. At this point, the attention strategy and currently executing behaviour are switched from exploration mode[9] (the state they were in during Stage 1) to task execution mode (using the newly combined behaviour created in Stage 2).

By practising its newly updated skill, the agent will hope to learn two things:

1. Whether the new behaviour is an improvement upon the old (trivially true if this is the first learning episode).

2. Where in task space any weaknesses in the new behaviour lie.

Both of these require the agent to receive feedback on its performance from sources that can both observe the agent's demonstration, and have knowledge of what (they think) the task requires. This could apply to the agent itself, to other agents sharing the task environment (e.g. a coach or team-mate), and to passive agents observing from outside the environment (e.g. a commentator or critic). In our framework, such knowledge is represented by behavioural *error metrics* and (possibly) task *goal states*. Performance feedback is received via an error signal

[9]Or from whichever behaviour was being executed if others have been used in the mean time.

generated by applying the metrics to the observed demonstration. The idea is similar in essence to the *actor-critic* method for Reinforcement Learning (Witten, 1977; Barto et al., 1983). We first look in detail at the different forms error metrics can take, then at how the error signals generated can be used for subsequent learning.

3.3.1 Error Metrics and Correspondence

Our approach of using behavioural error metrics to assess task performance is an adaptation of an approach used in imitation learning research; in particular, research into the *correspondence problem*. By far the most work in this area has been done by the Hertfordshire Adaptive Systems Research Group, therefore we give their definition here:

> "Given an observed behaviour of the model, which from a given starting state leads the model through a sequence (or hierarchy [or program]) of sub-goals in states, actions and/or effects, one must find and execute a sequence of actions using one's own (possibly dissimilar) embodiment, which from a corresponding starting state, leads through corresponding sub-goals — in corresponding states, actions, and/or effects, while possibly responding to corresponding events."
> (Nehaniv and Dautenhahn, 2002, p.43)

Solving the correspondence problem requires solving another: that of assessing the quality of correspondence between an observed behaviour and one's own attempt to imitate it. This, in fact, is very similar to the problem under current consideration: assessing the quality of correspondence between an observed behaviour and some assessor agent's definition of a target behaviour for a given task. Nehaniv and Dautenhahn (2001) define correspondence metrics in terms of their formal definitions of states, action-events and correspondences. Elsewhere they describe three levels of imitative correspondence: the *action-level*, the *program-level*, and the *effect-level* (Nehaniv and Dautenhahn, 1998). We now give an interpretation of these categories from a task learning perspective, and construct example error metrics for each in terms of the components of our framework.

Action-Level Metrics

Action-level imitation usually describes the process of acquiring novel movement primitives through the (relatively) precise reproduction of observed motion. Much of the foundational biological imitation literature focuses on this level, and where imitation is defined, it is often defined in these terms.

An action-level task error metric, then, would serve to detect deviations from a continuous motion path. Since task error is judged from an observer's point-of-view, a continuous motion path corresponds to a continuous path through sensor space. This also allows the concept of motion to have meaning in virtual reality domains, for example. In our framework, the mapping of continuous sensor space onto continuous motor space only happens within the lowest level of action elements. In other words, the only conceivable use for such an error signal would be to alert the agent that one of its action elements was inadequate. Correcting such a fault, and learning action-level tasks in general, is outside of the scope of GTLF – this is discussed further in Section 10.3. This statement comes with a proviso, however: some apparently action-level tasks could be represented in GTLF by using a finer-grained set of action primitives (see below).

For the sake of completeness, an action-level error metric would be a function of the form:

$$d_\alpha : \sigma \times \sigma \to \mathbb{R} \tag{3.15}$$

where σ is the set of all sensor states. At any point during testing, the output error signal is equal to $d_\alpha(\sigma_O, \sigma_E)$, where σ_O is the observed state of the demonstrator, and σ_E is the expected state. For example, suppose we have an agent that inhabits a 2D Cartesian plane, and has two sensors; one to detect its x-co-ordinate and one to detect its y-co-ordinate. It's actuators can propel it in any direction in the plane, and it's task is to move from $(0, 0)$ to $(0, 1)$ in a straight line. One possible action-level error metric would be:

$$d_\alpha(\sigma_O, \sigma_E) = d_\alpha((x_O, y_O), (x_E, y_E)) = |x_O| \tag{3.16}$$

that is, the magnitude of the agent's deviation from the y-axis.

Program-Level Metrics

In contrast to the action-level, imitation at the program-level involves acquiring novel behavioural *structure* from observation, using *existing* action and perception primitives. The detail of low-level movement need not be accurately reproduced, and could in fact be quite different. Byrne and Russon (1998) coined the term 'program-level' in part to establish the existence of a kind of imitation that *wasn't* action-level, and even go so far as to suggest that most (biological) examples of action-level imitation could be considered as instances of fine-grained program-level imitation[10].

We interpret task performance assessment at the program-level as being equivalent to identifying mismatches in perception-action association. Put another way, we are still interested in *how* the task is achieved, but only at the level of function rather than form (action-level). A program-level metric, therefore, has the form:

$$d_\pi : S \times S \to \mathbb{R} \tag{3.17}$$

where S is the set of all skill functions. As an initial simplifying assumption, let us assume that the observer and demonstrator share the same perception-action categorisation (trivially true if the agent is also the observer). Then we propose the following definition:

$$d_\pi(s_O, s_E) = \sum_{P_n \in \mathcal{P}(P)} d_A(s_O(P_n), s_E(P_n)) \tag{3.18}$$

where s_O is the demonstrator's skill function, s_E is the skill function expected by the observer, P is the set of all perceptual classes (common to both agents), $\mathcal{P}(P)$ is the power set of P, and $d_A : A \times A \to \mathbb{R}$ is a *metric on action elements*. A simple example of such a metric would be:

$$d_A(a_O, a_E) = \begin{cases} 0 & \text{if} \quad a_O = a_E \\ 1 & \text{if} \quad a_O \neq a_E \end{cases} \tag{3.19}$$

Under this arrangement, the error value is basically the number of mismatched

[10]They see individual muscle contractions as the only 'true' movement primitives, and argue anything which requires more co-ordination could be viewed as a 'program' of such contractions.

perception-action associations. This value can be normalised:

$$\bar{d}_\pi(s_O, s_E) = \frac{d_\pi(s_O, s_E)}{|\mathcal{P}(P)|.||d_A||_\infty} \tag{3.20}$$

where $||.||_\infty$ is the uniform norm[11]: $||d_A||_\infty = \sup\{|d_A(a_i, a_j)| : a_i, a_j \in A\}$. For the above example of d_A, the denominator equals $|\mathcal{P}(P)| = 2^{|P|}$, since $||d_A||_\infty = 1$.

These metrics assume that the skill function of the demonstrator is directly accessible, and therefore make an empirical testing stage redundant. If, however, the perception-action associations must be estimated from a series of observations $\Omega = \{(P_1, a_1), (P_2, a_2), \ldots, (P_N, a_N)\}$, then Equation 3.18 becomes:

$$d_\pi(s_O, s_E) = \sum_{1 \leq n \leq N} d_A(a_n, s_E(P_n)) \tag{3.21}$$

and in Equation 3.20, $|\mathcal{P}(P)|$ is replaced by $|\Omega|$. This can provide an estimate of the true error value after each observation, and is clearly affected by the breadth of perceptual states observed. To limit the effects of noise at the beginning of the demonstration, the normalised metric could be weighted as follows:

$$\bar{d}'_\pi(s_O, s_E) = \alpha + (1 - \alpha)\bar{d}_\pi(s_O, s_E) \tag{3.22}$$

where $\alpha = \frac{1}{n}$, and n is the number of observations made so far. In other words, the more observations made, the more confidence is put in the estimated error value.

The same technique can be used even if the demonstrator's perceptual categorisation is unknown. In this case the perceptual states, P_n in Equation 3.21, are defined according to the *observer's* categorisation. Now, the perceptual states of the demonstrator and observer do not necessarily co-incide. Therefore, observing both (P_1, a_1) and (P_1, a_2), for example, does not necessarily imply that the demonstrator has acted inconsistently with its own policy; it may just mean that P_1 overlaps more than one demonstrator perceptual state. If, furthermore, the action repertoires of the two agents differ, then the metric on action elements, d_A, must define the relationship between those of the demonstrator and observer. An example of a program-level behaviour metric in use is given in Section 5.4.

[11]NB. equation 3.20 is only well-defined if d_A is bounded, and thus $||d_A||_\infty < \infty$.

It is assumed that the demonstrator (that is, the learner) does not have access to s_E (that is, the expected behaviour) or else it would have nothing to learn. Program-level metrics are designed for use by an external assessor. Knowledge that the learner may have, however, relates to the desired *goals* or *effects* of the task — this is the third and final level of task assessment.

Effect-Level Metrics

Effect-level imitation describes the process of forming behaviour which will achieve similar 'results' to those achieved by an observed behaviour (Demiris and Hayes, 1997). Neither low-level movement nor behavioural structure need necessarily be reproduced, as long as the relevant goals are met.

Effect-level task performance, therefore, is determined by the states the task environment goes through, in relation to the goals required by the task. In our framework, both of these are described by perceptual states, so an effect-level error metric is of the form:

$$d_\epsilon : \mathcal{P}^2(P) \times \mathcal{P}^2(P) \to \mathbb{R} \tag{3.23}$$

where P is the set of perceptual classes discernible by the observer (i.e. assessor), $\mathcal{P}(P)$ is the set of all perceptual states discernible by the observer, and thus $\mathcal{P}^2(P)$ (that is, $\mathcal{P}(\mathcal{P}(P))$) is the set of all subsets of perceptual states.

To illustrate, let $G \in \mathcal{P}^2(P)$ be the set of all goal states for a given task (according to some task assessor), and $T \in \mathcal{P}^2(P)$ be the *trace* of environmental states observed during a task demonstration (from the perspective of that assessor). To simplify, suppose also that the task has a single goal state, $G_1 \in \mathcal{P}(P)$, so that $G = \{G_1\}$. We could define the following metric:

$$d_\epsilon(G, T) = \inf\{d_P(G_1, T_n) : T_n \in T\} \tag{3.24}$$

where $d_P : \mathcal{P}(P) \times \mathcal{P}(P) \to \mathbb{R}$ is a *metric on perceptual states*[12]. A simple

[12]Recall that the *infimum* of a set X of real numbers, denoted $\inf\{X\}$, is defined as the *greatest lower bound* of that set. That is, it is equal to the greatest real number that is less than or equal to all the members of X.

example of such a metric would (as before) be:

$$d_P(G_1, T_n) = \begin{cases} 0 & \text{if} \quad G_1 = T_n \\ 1 & \text{if} \quad G_1 \neq T_n \end{cases} \tag{3.25}$$

This arrangement gives an error of 0 if the exact task goal state G_1 was visited at any time during the demonstration, and an error of 1 otherwise. Another possibility for d_P is

$$d_P(G_1, T_n) = \begin{cases} 0 & \text{if} \quad G_1 \subset T_n \\ 1 & \text{if} \quad G_1 \not\subset T_n \end{cases} \tag{3.26}$$

which allows the goal state to be *included in*, rather than exactly equal to, one of the visited states. This is useful if the goal is only defined on part of perceptual space. To generalise to multiple goals which can be visited in any order, Equation 3.24 becomes

$$d_\epsilon(G, T) = \sum_{G_m \in G} \inf\{d_P(G_m, T_n) : T_n \in T\} \tag{3.27}$$

which can be normalised by dividing by the number of goals:

$$\bar{d}_\epsilon(G, T) = \frac{d_\epsilon(G, T)}{|G|} \tag{3.28}$$

This gives a final error value of $1 - \frac{n}{|G|}$, dependent only upon $0 \leq n \leq |G|$, the number of achieved goals. Even though we may not care *how* task goals are achieved, there may still be other motivating factors that we would like our metric to capture, such as how fast or how efficiently they are achieved. In this case, we are interested in the *change* in error value with respect to the variable we care about. If we call this variable x, this can be written as:

$$\frac{\Delta \bar{d}_\epsilon}{\Delta x} \tag{3.29}$$

We would like the *greatest possible decrease* in error for the *least possible increase* in x. Therefore we are seeking to *minimise* this quantity, and it can be viewed as another error metric. For example, suppose two different behaviours lead to the satisfaction of all goals in a task, but behaviour A takes 60 seconds and behaviour B takes 90. So we have $\Delta \bar{d}_\epsilon(G, T_A) = \Delta \bar{d}_\epsilon(G, T_B) = -1$, $\Delta t_A = 60$,

and $\Delta t_B = 90$. So

$$\bar{d}^t_\epsilon(G, T_A) = \frac{\Delta \bar{d}_\epsilon(G, T_A)}{\Delta t_A} = -\frac{1}{60} \; < \; \bar{d}^t_\epsilon(G, T_B) = \frac{\Delta \bar{d}_\epsilon(G, T_B)}{\Delta t_B} = -\frac{1}{90} \quad (3.30)$$

i.e. behaviour A produces a lower error value. This could equally well be applied to distance travelled, number of operations carried out, fuel expended, etc.

We consider two final cases: firstly, tasks which require goals to be executed in a particular *sequence*. Here, we need $G = \{G_1, G_2, \ldots, G_N\}$ and $T = \{T_1, T_2, \ldots, T_N\}$ to be temporally *ordered sets*. The case in which failing to achieve a certain goal G_n *prevents* the agent from achieving goals $G_{n+1}, \ldots G_N$ is already covered by the metric given above, which just penalises the agent for uncompleted goals. We are left, then, with the case in which it is possible but undesirable to complete goals in the wrong order. One way of doing this would be to alter the metric on perceptual states as follows:

$$d_P(G_m, T_n) = \begin{cases} 0 & \text{if } G_m \subset T_n, \text{ and } \forall\ i < m\ \exists\ j < n \text{ s.t. } G_i \subset T_j \\ 1 & \text{otherwise} \end{cases} \quad (3.31)$$

In this example the ordering is very strict; if any goal in the sequence is missed, all subsequent completed goals are treated as unsatisfied.

Lastly, for some tasks, it may be desirable for an error metric to reflect that an agent can get 'close to' a goal that is never actually attained, and that this is preferable to getting 'nowhere near' it. Again, this could just involve using a 'softer' metric on perceptual states, and may be best illustrated by an example. Suppose the task of collecting tokens from an arena is set, and let P be the set of perceptual classes that our (self-assessing) agent can discern, G be the set of goal states, and T be an initial task trace:
$P = \{$`many_tokens_left`, `some_tokens_left`, `few_tokens_left`, `no_tokens_left`$\}$
$G = \{\{$`no_tokens_left`$\}\}$
$T = \{\{$`many_tokens_left`$\}, \{$`some_tokens_left`$\}\}$

We can allow the agent's moderate success to be reflected by defining the

following metric on perceptual states:

$$d_P(\{\texttt{no\_tokens\_left}\}, T_n) = \begin{cases} 0 & \text{if } T_n = \{\texttt{no\_tokens\_left}\} \\ \frac{1}{3} & \text{if } T_n = \{\texttt{few\_tokens\_left}\} \\ \frac{2}{3} & \text{if } T_n = \{\texttt{some\_tokens\_left}\} \\ 1 & \text{if } T_n = \{\texttt{many\_tokens\_left}\} \end{cases} \tag{3.32}$$

and so $\bar{d}_\epsilon(G, T) = \frac{2}{3}$.

3.3.2 Making Use of Error Feedback

It may be that during or after a demonstration, error feedback is transmitted from multiple sources simultaneously, in which case the agent would need a system for weighing and integrating such feedback appropriately. For example, the more advanced the task knowledge of a critic, the more weight should be given to the error signal they transmit. If the error signals received convince the learning agent that the new task behaviour is an improvement upon the old, then it can be copied into long-term memory (along with its error estimate), and the old discarded. Less straightforward is the process of interpreting and storing error data for later use in the perception-action reconfiguration stage (see Section 3.4), and we look at this now.

Recall that we assume program-level error signals necessarily originate from an external source. While d_π as defined in Equation 3.21 can provide a measure of overall skill performance, the metric on action elements, d_A (Equation 3.19), can be more useful for identifying particular regions of task error. For example, if an assessor observes the demonstrator carry out action a_n in state P_n, then

$$d_A(a_n, s_E(P_n)) \tag{3.33}$$

where s_E is the expected behaviour, gives the assessor's error measure for that particular action. If the error signal generated by d_A can be transmitted to the demonstrator, then perceptual categorisation data can be generated for use by the reconfiguration module (see below). Such a datum would be of the form:

$$(P_n, w_n)_c \tag{3.34}$$

indicating a possible behavioural flaw in state P_n. Accompanying weight w_n could be used to indicate the learner's confidence in the error signal it has received.

It is more difficult, as one would expect, to identify the location of flaws in behaviour based only on information about goal states. For this reason, task learning is hard even for an agent who has set themselves the task. The non-satisfaction of a goal or goals, as indicated by a d_ϵ metric (Equation 3.23), implies behavioural flaws, but in which perceptual state? If there are no other constraints on the goals then this may be impossible to discern. However, suppose that the goals must be reached in sequence $G_1, G_2, \ldots, G_n$. If G_i is the first goal not to be satisfied, the perceptual states occupied *after* the completion of G_{i-1} are more likely to have generated faulty behaviour than others. These states can be found by examining the task trace between G_{i-1} and G_i (see above). If the learner is also an assessor (i.e. possesses an appropriate effect-level metric), then this information can be used to generate perceptual categorisation data directly. If the assessor is external, it must first be passed to the learner. There are, of course, many other possible constraints on goals which could inform this process.

There are a number of conditions which could bring an end to the testing phase, including:

- The task (or the opportunity to practise it) being removed from the environment.
- The agent reaching some threshold confidence level in the new behaviour, allowing it to make decisions regarding the extent of its improvement and / or the source of faulty behaviour.
- The period of time prescribed for testing by the agent elapsing.

In any of these cases, task interaction will cease and the agent will wait as before for an opportunity to apply its findings.

3.4 Stage 4: Reconfiguration

It may be that iterating the prior learning stages results in the agent converging upon a behaviour which completes the task to an acceptable standard. If, however, a performance ceiling is reached before this occurs, then it may be that the

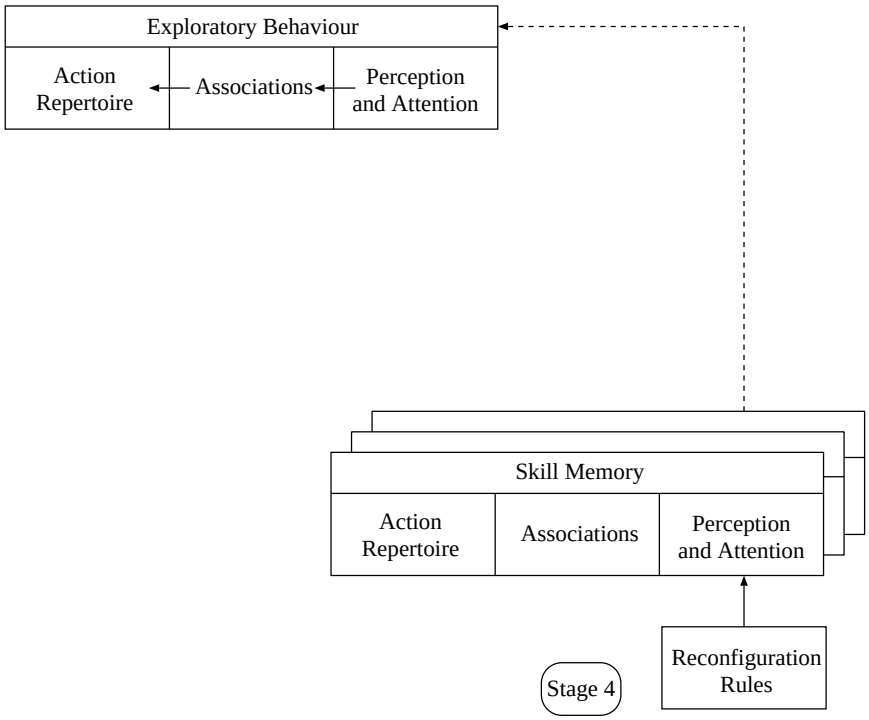

Figure 3-6: Stage 4: Reconfiguration — if the prior learning stages consistently fail to improve task performance, it may be possible for the agent to reconfigure its perception-action primitives.

agent's perception-action representations themselves are inadequate to define a solution. In this case, some agents may be able to reconfigure these primitives so that learning can continue.

This process can be contrasted with Stage 2 as follows: during consolidation, the perception/action representations were fixed, and the links and priorities were updated. This time the situation is reversed, and the purpose is to re-factor the representations being used to better fit the task while fixing the metadata values. The 'wake-sleep' algorithm for finding optimal internal representations in neural networks (Hinton et al., 1995) may contain helpful analogies here. In wake-sleep, bottom-up 'recognition' weights determine the internal (i.e. hidden unit) representation given an input, and top-down 'generative' weights estimate the input given an internal representation. In the 'wake' phase, recognition weights are fixed while generative weights are updated; analogous to our consolidation stage. The opposite occurs in the 'sleep' phase, which parallels this reconfiguration stage.

Reconfiguration requires reasoning about the perceptual categorisation data

both stored in the episodic buffer, and input from Stage 3. We can, without significant loss of generality, describe the agent's reasoning capabilities by a set of production rules which are satisfiable by criteria on the data and cause changes in mental representation (effectively executing 'mental actions'). By this we do not intend to make any implications about conscious control of the process, even if that term was well-defined for all agents. We return to our token-collecting example to illustrate.

Suppose this time that the tokens distributed around the area come in two colours; green and red; and that the task is to collect only green tokens. Suppose also that the agent has a visual sensor which generates RGB values for the perceptual system. At the start of this learning cycle, the agent could discern two colour classes defined in terms of this sensor value: `light` (RGB mean ≥ 127.5) and `dark` (RGB mean < 127.5). The red tokens have a mean RGB value (ignoring lighting effects) of 85, and the green tokens have a mean value of 68. Both these types of tokens are thus indistinguishable (`dark`) under this perceptual configuration. We assume that the agent's tendency to make errors when in the {`dark`} state (i.e. to mistakenly pick up red tokens) has been identified by an assessor during testing, and that the following datum has arrived here for processing:

$$(\{\texttt{dark}\}, -0.8)_c \tag{3.35}$$

The reconfiguration module has two options at this point. Firstly, if this a new mistake (i.e. it hasn't received this datum before), then it is quite possible that a change in skill mapping could rectify the problem. In this case the module will attempt to use its knowledge base to alter exploratory behaviour associations, so that the offending perceptual state will be more thoroughly explored during the next learning episode. If, however, this is a repeated mistake, then the module may try to alter the agent's perceptual configuration instead.

A simple rule which could be applied here would be to subdivide the perceptual class `dark` into two. Of course, this rule could take a number of cycles to achieve a reasonable separation between the red and green tokens, because only the mean value is being used. Also, a number of redundant classes will be created, potentially adversely affecting efficiency (see Section 2.3.3). Another rule could be used to merge redundant classes once error has been sufficiently

reduced. Alternatively, a more complex rule involving the separate consideration of R, G and B values could reduce error in fewer cycles.

The example above focuses on altering perceptual structure, but the principles could equally well apply to a task which requires, say, finer-grained action elements. We have shown elsewhere (Section 2.3.3) that when increasing representational complexity it is always possible, in theory, to at least *maintain* accuracy for a given task. Also note that although the agent was not initially differentiating between red and green, it was always *capable* of doing so. If instead the agent were equipped only with a binary light/dark sensor, it would never be able to make the necessary distinction, regardless of perceptual configuration. The agent would have reached *maximum perceptual resolution* in this region of task space, and no further performance improvements would be possible (in this region).

Agents implementing this module could vary greatly in their ability to improve their representations, due to the corresponding potential variety in the complexity of the rules which govern the process. The rules must be part of prior knowledge, and some agents may not possess them at all, effectively limiting them to their innate categorisations. At the other extreme, cognitively advanced agents (such as humans) may be able to consciously alter their primitives, e.g. "In the future I must remember x as an important special case of X."

Having made use of all available data for representational improvement, the learning cycle ends and episodic memory is cleared ready for the next cycle. How soon this occurs depends, amongst other things, upon when the task next presents itself.

3.4.1 Incremental Learning

It should be noted that, although episodic learning has advantages (see Section 2.3.3), some scenarios may require more rapid, online skill updates. In this case, the learning phase (Stage 1) could output a single observation, which could then be immediately 'consolidated' (Stage 2) with prior knowledge. If we make the reasonable assumption that the learning agent will at some point wish to put into practise what it has learned, then it is at this point we enter Stage 3 (Testing). If necessary, re-factoring (Stage 4) would occur before any more learning

takes place (if any more is possible). In other words, rather than the 1-2-3-4 episodic pattern, we have a 1-2-1-2-...-1-2-3-4 incremental pattern. In this case we say GTLF is operating in *incremental learning mode*.

3.5 Exploiting Task Structure

So far we have implicitly accounted for the acquisition of single, 'flat', reactive task behaviours. We wish, however, the framework to be sufficiently general to be able to handle tasks containing temporal sequences, and also those which can be hierarchically organised into sub-tasks.

3.5.1 Hierarchy

As previously explained, we wish to make as few assumptions as possible about the internal representation of our task behaviours, allowing them to be implemented in many different ways. Our only constraint is that they can in some way be interpreted as a map from sets of perceptual classes to action elements. Rather than the low-level problem of acquiring a single task behaviour, which would involve associating relatively high granularity perceptual classes with action elements, let us consider the problem of arbitrating *between* a set of known task behaviours. We refer to this as an *arbitration behaviour*.

Since an arbitration behaviour must be capable of deciding which task behaviour to execute, its perceptual classes must be high-level and coarse-grained. For example, suppose task behaviours X and Y have been learned, then the perceptual classes of the arbitration behaviour must allow it to determine whether each task is available in the environment for execution. The association structure (tabular, functional, etc.) must then determine which task it is appropriate to execute, or even if both can be executed concurrently. Once this has been decided, the action - or task behaviour in this case - is executed. Control passes down the hierarchy and the process continues. For highly complex tasks with many nested sub-tasks, it may be necessary to extend this hierarchical control system to many levels. In this case, we can borrow the slip-stack approach from Bryson's POSH reactive plans (Bryson, 2001): the top-level arbitration behaviour (analogous to the drive collection) and the *lowest level* task behaviour receive perceptual input

and can regulate control. The intermediate-level behaviours are not queried at every perceptual cycle, so that efficient traversal of the tree can be maintained.

Clearly, attempting to learn multiple levels of hierarchy simultaneously could potentially require enormous resources in terms of prior knowledge or learning (or both). However, if we assume a bottom-up approach in which all sub-task behaviours below the one in question have already been learned (to a sufficient degree), then there is no reason to consider learning an arbitration behaviour as requiring any different process to learning a task behaviour. Therefore, we can generalise the GTLF to hierarchy by simply making the assumption that the actions being associated during learning could include complete sub-task behaviours or skills.

3.5.2 Sequence

It is even more straightforward to deal with temporal sequencing in GTLF. Until now we have assumed that the perceptual state, which specifies a set of occupied perceptual classes, is sufficient to specify action. A temporal sequence, on the other hand, requires dependency on the previous action taken. Suppose we add to our categorisation perceptual classes pertaining to the state of the previous action taken — effectively a sensor sensing the past. Then, associations built between perceptual classes in this channel and actions specify a temporal sequence. So, as for hierarchy, we can use exactly the same process for learning temporal sequences as we do for any other type of task dependency. In fact, since we can in principle add an arbitrary number of such history sensors (constrained in practise by resources available), it is theoretically possible for GTLF to handle arbitrarily long temporal dependencies.

3.5.3 Summary

Based on the principles set out in the Chapter 2, we have specified a General Task Learning Framework which comprises four stages. In Stage 1, the agent explores the task environment according to the learning methods that are available, and that it wishes to use. Insight involves applying known rules and skill elements to the task; trial-and-error involves interacting with the task; observation involves watching an expert complete the task; and instruction involves attending to and

interpreting a teacher. In Stage 2, knowledge gained during exploration is combined with old to create an updated skill and attention strategy. In Stage 3, the new skill is tested and feedback received from any agents monitoring the learner's performance. In Stage 4, the learner's perception space is reconfigured to enable improved accuracy in future episodes, and actions are compounded where possible to improve efficiency. The new skill is stored and the cycle iterates. Some tasks may have inherent hierarchical or sequential structure which can be exploited to improve the efficiency of this process.

In Chapter 7, we describe in detail an implemented example of GTLF, together with the results of an experiment which demonstrate its use, before comparing and contrasting the framework with similar existing systems. Before that, however, we look at the bigger picture: what are the design lessons that could be learned from GTLF, and what are the practical implications for an agent designer?

Chapter 4

GTLF as a Design Philosophy

The aim of this chapter is twofold: to distill and summarise what we consider to be the main lessons in agent design to be learned from GTLF, and to discuss some of the more philosophical ideas and questions that have arisen from its creation. It is particularly suited to the busy designer who is interested in a quick 'take-home message' without the need to trawl through the technicalities of the longer chapters.

Before we begin the discussion proper, we make two general points. Firstly, above all else GTLF should be *useful*. If it is a hindrance rather than a help in designing or thinking about agents then it has failed in its primary purpose. We do not see it as a collection of rules or parameters that must be adhered to in order to create some kind of 'GTLF-compatible agent'. Rather, we hope GTLF is used as a collection of guidelines and structures, open to debate and revision, around which designers can organise their own ideas and research interests. If the reader considers any of the points put forward in this chapter before designing another agent, then we see that reader as using GTLF.

Secondly, although there is much advice on agent design contained within this chapter, it does not come from one who has vast experience and has designed countless agents. Some may suggest that this invalidates the advice, but we would argue that the perspective of the young researcher has advantages in this case. The author is not yet rooted in one particular way of thinking; is not allied to any specific academic cause or ideal; has a perhaps naïve yet still relatively fresh outlook on the subject. No doubt many established designers will have considered a number of the issues raised in this chapter, quite possibly more formally and

in more detail than given here. However, by collecting all such ideas in the same place, it is our hope that at least one will be novel to the reader.

4.1 Five Design Considerations from GTLF

There are a number of points that could have been added to the list below, but for the sake of brevity we have selected what we see as the top five. It is worth reiterating that these considerations are the product of the *entire* dissertation experience; if the chronology is not taken into account, it could be construed that the author does not follow his own advice. Put another way, the author learned some of these lessons 'the hard way'.

4.1.1 Consider the Whole Problem

It is tempting and, it seems, fairly common to present a task learning problem as consisting of just the part that needs learning plus whatever is needed to make that part comprehensible. The remainder is either swept neatly under the carpet or else simply ignored or taken for granted. This makes some sense when it comes to publishing work, as the alternative would be very lengthy papers with relatively little interesting content. However, we believe that it is important for the one actually carrying out the work to consider the whole problem.

For example, every problem contains hard-coding. From the physical design of the agent hardware (if applicable), to the operating system they are running (on), to the programming language being used, to the statistical assumptions being made, to the representations and algorithms which have been chosen, to the choice of algorithm parameters, and so on; without hard-coding there is no problem. It is unrealistic to expect everything to be learnable (at least not everything simultaneously) — there is no free lunch — so the best alternative is to try and define as precisely as possible what needs to be hard-coded or assumed. Once this is acknowledged, it should be easier to accurately define exactly what the core learning problem is.

Included within the rather broad definition of hard-coding given above is the definition of the task itself, and the (often implicit) metric for success associated with it. But it is worth asking where these quite fundamental definitions come

from. We assert that they do not exist independently of an agent. If (as is likely) the answer is "from the agent designer", then how tailored is the task and its success metric to the specific agent and environment which one hopes will generate (publishable) results? How much have the designer's social priors affected these definitions? Or perhaps the task definition originates from another agent, either external or internal to the experimental arena. How trustworthy is that agent? How high or strict a standard is expected? If the origin is the learning agent itself, does it fully understand the task, and can it reliably chart its progress? What if there are multiple definitions of the same task? How can a concensus be reached? How should different opinions be weighted? By asking and answering some of these questions, an agent designer may discover things they never knew they had assumed.

4.1.2 Consider Your Primitives

In some respects, to "Consider the Whole Problem" covers all possible bases. The remaining considerations listed here, then, are those that we feel are worthy of particular attention. As far as perception-action primitives are concerned, we suggest asking the following three questions:

1. *Where are my primitives coming from?* — this is part of the same question we asked above. Are the learning primitives for this problem hard-coded or themselves learned? Or perhaps more accurately, what *aspects* of my primitives are hard-coded? What silent priors have influenced those aspects?

2. *Where are my primitives going?* — are the primitives I have defined and / or derived for this problem static? If so, I'd better be pretty sure that they are suitable, but on the other hand not so tailored as to mask the difficulty of the problem in general. If it is anticipated that the perception-action primitives should change over time, then this is a very difficult learning problem in itself, orthogonal to the task learning problem in question. Can the two solutions evolve simultaneously, or is it better to have some kind of episodic 'E-M style' dithering between the two; that is, hold one problem fixed and optimise the other, and then swap over.

3. *What level of learning am I looking at?* — is it closer to the action-level

end of the spectrum, alongside robot controllers, where the primitives relate to quantities such as joint angles and motion paths? Or is it closer to the effect-level end, where the primitives are perhaps conflicting high-level goals and the various skills needed to achieve them. Or is it somewhere in between? Do the primitives themselves relate to each other independently of the particular task context? For example, can the perceptual primitives be nested or organised in such a way as to simplify the decision problem? Can low-level action primitives be concatenated and co-ordinated to create complete movements and behavioural units which can then in turn be applied to learning at a higher level?

4.1.3 Consider Different Learning Methods

It stands to reason that every learning researcher will have their favoured learning method, algorithm, paradigm, etc.; let's call it X. However, this can have the unfortunate side-effect that every learning problem is seen (at least initially) in terms of X. At worst, problems become hammered into type X problems, whether it suits them or not.

We are not proposing that all methods are equal; instead we suggest considering what parts of a given problem or what classes of problem in general might be suited to different learning methods. If we take as a case in point the two paradigms most familiar to us: social learning via imitation and individual learning in the form of Reinforcement Learning. The latter is suited to tasks which can be safely repeated many times over until some convergence criterion is satisfied; perhaps in simulation, for example. For RL to work, the agent must be able to choose its path through the task using some action selection scheme designed to maximise learning. Once a policy has been learned in this way, it could be used to adaptively control an agent which perhaps does *not* have the opportunity to repeat trials ad infinitum. Coupled with the proven convergence properties of many RL systems (MDPs for example), this is a very powerful method with proven real-world applications.

However, there are also many situations where it is impossible to use RL. Some tasks must be learned very quickly; those that result in death in the event of a bad choice for example. For any agent with limited energy, longevity, or

even just attention span, repeating tasks indefinitely may not be possible. On the other hand, social learning requires some kind of model agent to be available to learn from, and this immediately limits the situations in which it could be used. It is also very different in process from RL: the source of task knowledge is another (presumably) autonomous agent over which the learning agent exercises no control. The path through task space is determined not by the learner but by the demonstrator; 'random search' or indeed any control policy for the purposes of learning becomes meaningless. Also, the opportunity for learning to convergence is completely dictated by the demonstrator. If the demonstrations cease, then learning ceases, no matter how short the learning time may have been. In imitation learning you get what you're given, which is why so-called 'one-shot' learning is of critical importance. But by harnessing social knowledge in this way, some tasks which would take a huge amount of time to learn using RL could be learned very quickly.

It is also worth considering when different learning methods could be used in tandem or in sequence to complement each other. For example, using social learning when a model is available to garner 'rough' task knowledge as quickly as possible, and then using reinforcement learning when no model is available (or everything possible has been learned from it) to fine-tune behaviour to the agent's particular configuration or situation.

4.1.4 Consider Other Agents

Suppose that an agent designer designs an agent to learn and execute a given task in a given environment. What may not be apparent to the designer is that that agent represents a point in a huge design space of possible agents. Part of the role of GTLF is to try to define some of the 'dimensions' of that design space, and in doing so make the space easier to explore systematically. This could be useful if, say, the designer wants to find a better agent, as an alternative to just implementing a bunch of agents having somewhat random or 'best-guess' properties and parameters, hoping to chance upon an improvement.

This way of thinking about design spaces should also be of benefit when working with or reading about agents designed by others. It should more readily allow a designer to see how his agent relates to another, and encourages an open

mind when it comes to design differences: "I have done it this way out of these possible ways for these reasons, they have done it that way out of those possible ways for those probable reasons" as opposed to, "I have done it right, they have done it wrong."

The third and final point we would like to make in this section is to suggest that apparently quite different agents might lie closer to each other in design space than one might initially suppose. The particular hypothesis expounded later in this dissertation is that virtual agents have more in common with robotic agents in terms of embodiment than some researchers give them credit for. In other words, it could be that a particular agent embodied in a virtual domain could be 're-embodied' in a material domain without having to move very far in design space. It is worth noting that in general, this space is very likely not as well-behaved as we have made it sound.

4.1.5 Consider the Bigger Picture

As a researcher in X, you may only really be interested in one type of agent, one type of task, or even just one type of algorithm. However we would urge you to consider the 'bigger picture'. For example:

- Your agent might only ever need to learn one type of task, but what if it needed to learn many? How would it structure and co-ordinate that knowledge? How could learning be consolidated across multiple episodes and be extended to lifelong learning?

- Your agent may be required only to operate in a constrained environment, but suppose its sphere of operation needed to be widened. How could its task knowledge be adapted or transferred to new domains, both ostensibly similar and obviously different? Would this require major change at all levels of representation within the agent, or only at the levels close to the agent-environment interface? How agnostic is your learning system to its implementation platform?

- Your agent may only ever act in isolation, but how would it cope if forced to interact with other agents? Could it learn to co-operate? Could it learn to compete? Could it learn to defend itself? Could it learn to attack? How

could it be adapted to make use of socially available task knowledge? How could it be adapted to *impart* its own task knowledge socially?

- You might only be interested in the learning system of your agent, but what about the rest of the architecture? What other modules are there, how are they implemented, and where do they come from? How can your learning system best support the data requirements of other modules and vice versa?

Looking at some of these questions should allow both yourself and others to better place your work within the vast and expanding body of learning agent research.

4.2 Summary

By treating GTLF as a broad design philosophy as opposed to either an agent classification framework or baseline learning system, we find five considerations to put forward to the learning agent designer. Firstly, consider the whole problem, as opposed to just the fragment of interest; define what is hard-coded so it is easier to define what needs to be learned. Secondly, consider the primitives being used; where they come from and how they might change. Thirdly, consider using different learning methods; recognise their relative strengths and weaknesses in different learning scenarios. Fourthly, consider your agent as one taken from a large design space; try and relate your agent to others. Fifthly, consider the bigger picture; although you might be interested in only a relatively constrained problem, try and hypothesise how the agent's operation could be generalised.

This first part of the dissertation has concluded with a philosophical discussion on agent design, based upon the lessons learned by the author by the end of the project. In the next part we restart the story at the beginning, describing the origins of our interest in task learning, and documenting the evolution of the framework set out in this part.

Part II

System Development

Chapter 5

COIL: Cross-channel Observation and Imitation Learning

In Part I we took a theoretical path, looking first at the concepts that we believe are core to task learning, and then at our framework in detail with reference to examples and hypothetical implementations. In Part II we essentially take the same route, but from the more concrete, empirical perspective of chronological systems development. We trace the evolution of our framework via the systems and algorithms that have inspired us, toward the current GTLF implementation.

At the outset of this project, our primary interest was in human-level real-time social learning (Bryson and Wood, 2005; Wood, 2006). We were persuaded, chiefly by the arguments of Laird and van Lent (2001), that the domain of virtual reality-style computer games could offer a fresh perspective on this research topic, also allowing for relatively rapid development when compared with other domains (see Section 5.2). We settled on the particular problem of using program-level imitation (Byrne and Russon, 1998) to allow virtual agents called *bots*, which inhabit the game world of *Unreal Tournament* (Digital Extremes, 1999), to learn skills that might improve their in-game performance. In search of inspiration, we began to investigate other high-level, real-time social learning systems,

One such system which not only produced impressive practical results, but also comprised a computational model with convincing biological correlates, is Deb Roy's Cross-channel Early Lexical Learning (CELL) system (Roy, 1999; Roy and Pentland, 2002). CELL essentially learns to associate audio input produced by spoken words (e.g. "ball") with visual input which contains the referent(s)

of those words (e.g. a picture of a ball). As we studied CELL more closely, it became apparent that the process it used to learn bindings between spoken words and their perceived meanings might be adapted to learn bindings between more general percepts and actions, as observed during the demonstration of a task. The further we explored how such a generalisation might be made, the more we started to learn about and question the components and processes necessary for learning which were task- and / or agent-independent. This, then, was the origin both of our interest in the nature of task learning, and of the sequence of systems which has (thus far) resulted in GTLF.

In Chapter 3 of his thesis dissertation, Roy gives an implementation-independent description of CELL, stating:

> "...[CELL] may be applied to a variety of domains beyond those implemented in this thesis." (Roy, 1999, p. 47)

We had three reasons to test this hypothesis:

1. To see if we had found / could create a system capable of performing program-level imitation in Unreal Tournament.

2. To see what could be discovered about task learning in general through the domain transfer process.

3. To test if our target domain was included in the 'variety of domains' conjectured by Roy.

This chapter explains our modification of CELL to create COIL – Cross-channel Observation and Imitation Learning – and presents experimental results using the modified system. We start by giving an overview of the workings of the original CELL model, and explain how it could in principle be adapted to program-level imitation. We then defend our original choice of Unreal Tournament as a research platform, and present some test tasks that an imitation system would ideally be able to learn from an appropriate demonstration. Next we go through each stage of COIL in detail, relating them back to CELL, and also explain our addition of a module for executing learned behaviour. Finally, we give the results of COIL's performance on the test tasks, which show some degree of imitative success. The chapter concludes by summarising what we had learned

from our initial COIL implementation, and introduces the subject of Chapter 6; a critical analysis of COIL.

5.1 CELL: a Working Learning System

As the name suggests, CELL was designed to emulate lexical acquisition in infants; specifically, to associate views of a given target object with spoken words describing that object. Roy's experimental setup consisted of a camera mounted on an articulated robot arm which moved around an object to capture it visually from different angles. The sounds it was trained on were the utterances of a series of parents during sessions of natural[1] interaction with their infants, and their joint interaction with the target objects of the experiment. This occurred independently of the picture data capture, and the parents were not informed that their interactions should achieve any specific goals. A noise-cancelling microphone was used, and the utterances were subsequently digitally sampled for processing. The association of words to pictures was carried out artificially by an analyst noting the period during which a certain object was the focus of attention, and then pairing that period of sound with pictures of that object.

The CELL learning model consists of five main stages (see Figure 5-2 and details below in Section 5.3):

1. *Feature Extraction* — salient features of the input sensor stream are extracted and isolated in separate Linguistic and Semantic data **channels**.

2. *Event Segmentation* — the channels are divided temporally into chunks called **events** and **subevents**.

3. *Co-occurrence Filtering* — Linguistic and Semantic chunks which co-occur are linked together and stored in a short-term memory buffer.

4. *Recurrence Filtering* — any paired chunks which are repeated in close temporal proximity are extracted and stored in a mid-term memory buffer. These pairs are **lexical candidates**.

[1] At least, as natural as possible given the circumstances of a controlled laboratory experiment.

5. *Mutual Information Filtering* — cross-channel Linguistic-Semantic mutual information is calculated for all lexical candidates. Those for which the value is sufficiently high are output (into a long-term memory buffer) as **lexical items**.

By the definitions we set out in Part I, CELL's language-learning method falls under non-imitative social learning – its task is just to *acquire* knowledge from observation. However, if we were to extend this task by asking it to *generate* spoken words given visual input (whether or not this behaviour is actually expressed), then this moves us toward imitation; toward COIL. In this extended context, we can think of CELL's perception space as consisting of all possible views, and its action space as containing all possible vocalisations (the converse is the case if we assume the task is to match objects to given speech). Perceptual classes, then, are groups of views which represent certain object properties, and action elements are (strings of) phonemes. These are extracted from camera and microphone data respectively through CELL's Feature Extraction and Event Segmentation stages, which thus define the perception and action categorisations available to subsequent learning processes. So, how does CELL solve the four task-learning sub-problems outlined at the start of Section 2.3? Co-occurrence Filtering creates bindings between perceptual classes and action elements, and Mutual Information Filtering assigns a real number; a measure of mutual information; to these bindings. This constitutes the agent's solution to sub-problem 1 (skill formation / improvement). Both of these stages, along with Recurrence Filtering, contain mechanisms for eliminating classes and elements which are unlikely to be relevant to the task at hand. This can be seen as forming the agent's attention strategy (sub-problem 2). Improving the perception and action configurations (sub-problems 3 and 4) would require progressively improving the rules encoded in the Feature Extraction and Event Segmentation stages. CELL does not include this feature.

Having recast CELL in terms of task learning by imitation, we can now attempt to generalise its domain of operation.

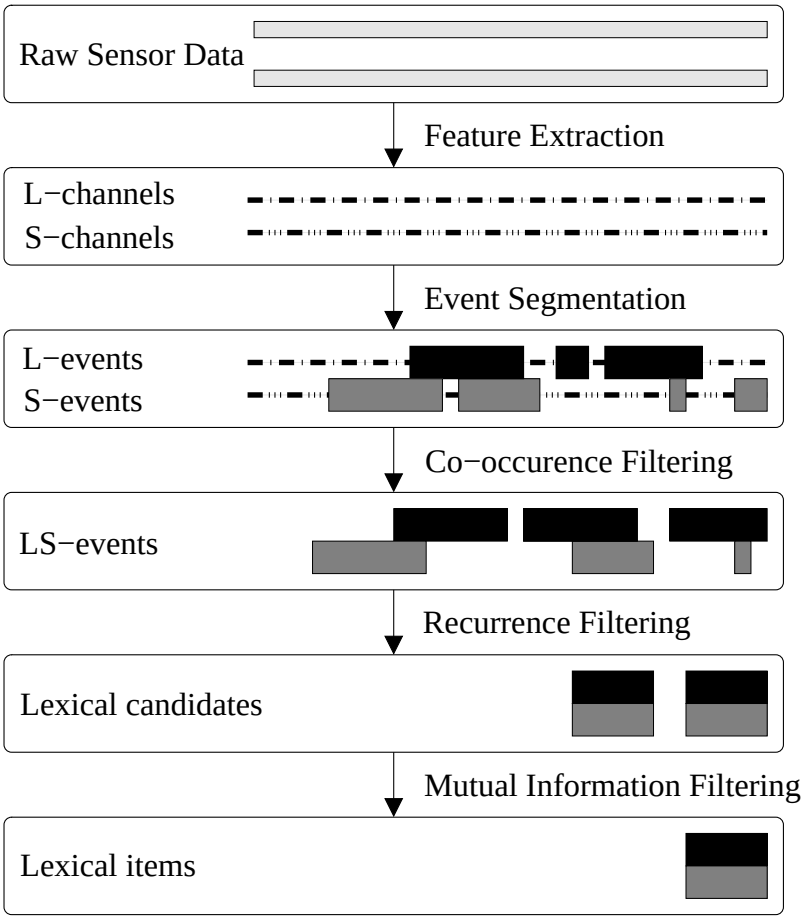

Figure 5-1: The inputs and outputs of each stage of CELL: *Feature Extraction* separates the raw sensor data into Linguistic and Semantic channels; *Event Segmentation* identifies discrete blocks of data within each channel set; *Co-occurrence Filtering* pairs co-occurring Linguistic and Semantic events; *Recurrence Filtering* identifies those pairs which have been seen most frequently; *Mutual Information Filtering* creates prototype pairs from those which yield the highest information (Roy, 1999).

5.2 Learning in Different Domains

We wish to move from CELL's domains of object recognition for perception and vocalisations for action into a more task-independent framework. As we have done throughout, we are considering situated agents of the sort described in Chapter 1. For imitation, this implies that an embodied imitator agent remotely observes an embodied expert agent acting in a shared environment. Human imitation, for example, is a special case which satisfies these constraints.

COIL's goal is to create executable skills – that is, bindings of perception to action – through imitation. Therefore, COIL replaces CELL's Linguistic and Semantic channels with generic Action and Perception channels. The Action channels receive data relating to the actions executed by the expert while completing the task. The Perception channels receive data relating to the perception of the expert, which is then used to determine the context of actions. Of course, the context determining actions includes internal state unlikely to be observable by the imitator, as well as external state which, while visible, will not perfectly correspond to the expert's view.

As Roy points out (Roy, 1999, ch. 2), human infants possess innate biases which make learning tractable (see also Thiessen and Saffran, 2003). This is reflected in CELL by, for example, the extraction of shape and colour characteristics from captured images, and the automatic recognition of phoneme boundaries. COIL is no different: the success of the Feature Extraction and Event Segmentation stages depend upon the imitator's biases to filter out extraneous sensor information and parse continuous behavioural / perceptual data streams into representative categories (Byrne, 2003). Thus, the data going into the 'main' processing stages of COIL (Co-occurrence, Recurrence and Mutual Information Filtering) consist of select, segmented Action and Perception channels. Details of these stages can be found below, but the key question for now is: what is the output of the model?

In high-level terms, the resultant chunks stored in COIL's long-term memory buffer represent actions paired with perceptions. Since the goal of an imitator is (presumably) to act, we can view these chunks as building blocks for a specification of imitated behaviour. If a given perception has been seen to instigate some action, this could be described as **motivation**. Conversely, if an executed action

has been seen to bring about some perceptual state, this could be called **expectation**. Hence the output of COIL's fifth stage consists of **Motivation Items** and **Expectation Items**, which I henceforth refer to collectively as **M-E Items**. These chunks are similar to Drescher's *context/action/result* schemas (Drescher, 1991), except that COIL does not go so far as to merge its observed *context/action* (Motivation) and *action/result* (Expectation) pairs. The issue of deciding how to act based upon M-E Items is beyond the scope of CELL's original operation, but, since it is such a crucial part of the imitation process, we have added extra stages to COIL specifically for this purpose (see Section 5.3.6). Note that, if the perceptual categorisation is correctly tailored so as to create mutually-exclusive classes, then a well-defined skill function can be formed without the need for an attention strategy. If, however, in order to be generally powerful and reusable, M-E Items *do* sometimes include non-exclusive perceptual state, then an attention strategy will have to be inferred.

In the next sections, we realise this abstract description of COIL. We first introduce our chosen task domain and the initial test behaviours to be imitated therein. This is followed by a detailed breakdown of each stage of the model, supported by implemented examples, and contrasted with examples from Roy's implementation.

5.2.1 The Real World, Robotics and Realistic Simulations

Much recent imitation research makes use of robots (or at least robot simulators) as the platform to test new models and algorithms (Alissandrakis et al., 2005; Hafner, 2005; Schaal, 1999). Robots have many advantages as experimental platforms: they operate in the real world, in real time and face many of the same problems as human imitators while being able to exploit similar constraints (e.g. the physics of gravity, optics and impact). Robots like humans must deal with noisy and incomplete sensor information, imperfect motor control, and dynamic, unpredictable surroundings.

On the other hand, many practical issues associated with robots can inhibit research. Robots may severely exaggerate the effects of noise, since it is difficult to tune them to the precision achievable by a human infant learning hand-eye coordination. Maintaining them takes considerable cost and expertise that is

orthogonal to artificial intelligence, often requiring special technicians in the laboratory. As a result of these constraints, robot tasks seldom approach anything like the complexity of animal behaviour. Animal behaviour is characterised by multiple, dynamic and conflicting goals in environments populated with agonistic agents[2]. It also is characterised by phased or cyclic activity operating on multiple concurrent time courses. Because robot behaviour is currently so severely constrained, concentrating all research on these platforms can result in overlooking combinatorial issues in learning and action selection that might be quite accessible in other platforms, such as artificial life (Tyrrell, 1993; Zettlemoyer et al., 2005).

For this reason, we have opted for what we believe is a 'best of both worlds' domain: real-time, virtual reality-style computer games. This is certainly not a new approach, and there are many AI researchers already working in this domain (Laird, 2001b; Le Hy et al., 2004; Thurau et al., 2004a; Gorman and Humphrys, 2005). Although games do avoid some of the technical problems of robots, they do introduce others. Again, perception and action may be far less reliable than for a skilled animal, and further they can be unreliable in ways that are bizarre by animal standards (e.g. failure to report the presence of a wall blocking movement.) However, they are real-time and highly dynamic environments which require the pursuit of multiple goals (defeating aggressive opponents, curing injuries, accumulating weapons and/or other tokens, rescuing innocents, assisting teammates). They contain both continuous and discrete actions and perceptions. Also, importantly, they are not constructed by the experimenters as biased "toy" domains (Bryson et al., 2001). Rather, they present tasks on a general-purpose platform that are challenging for even human intelligence.

We now describe the game we have chosen, *Unreal Tournament*, highlighting its suitability for our purposes.

5.2.2 Unreal Tournament

Unreal Tournament (UT) is a commercially released, multi-player 'First Person Shooter' (Digital Extremes, 1999). As the term suggests, the user has an agent's-eye view of the game and direct, real-time control of an avatar's actions. UT also

[2]By *agonistic* we mean competing for survival and therefore liable to conflict.

supports remote control of agents by sending commands to the game server over a network, which provides a framework for allowing external programs to direct an agent's actions. Thus, UT provides a viable platform for testing strong AI, since humans and AI can compete and interact in a real-time, spatially situated domain. AI-controlled agents are commonly known as **bots** in the literature and gaming community. The game server sends two categories of sensor data back to the client. The first is synchronous: at regular intervals the client is informed of the agent's status (e.g. health, ammunition, current weapon, etc). The second is asynchronous: for example whenever a wall is bumped, a footstep is heard or damage is taken.

Figure 5-2: A screenshot from the virtual reality-style computer game, *Unreal Tournament* (Digital Extremes, 1999).

These data, when viewed as a whole, are a highly dynamic, high-dimensional mixture of categorical and continuous variables, akin to sensory data acquired in the real world. Other similarities include physics simulation, such as collisions,

gravity, and sound and light transmission. The bots' sensor data are incomplete in the sense that only a reduced subset of the game variables are observable; the bots have limited virtual sensors. For example, the imitator cannot know the health state of the expert, although this may well affect the expert's choices. However, what sensors are available are not subject to noise in the same way physical sensors would be. Neither are bots hampered by imperfect motor control. In fact, low-level movement (i.e. that of arms or legs) is dealt with by the graphics engine, and bots can only be externally manipulated at a higher level (i.e. move forward or rotate). This may seem unrealistic to those familiar with fine gesture simulation (e.g. Schaal et al., 2003), but note that there is a good deal of neurological data indicating that the 'higher' level brain functions we are presumably simulating (e.g. frontal lobe control of behaviour) can also operate at a similar fairly gross abstract level (Bizzi et al., 1995; Graziano et al., 2002). Much of the natural imitation literature does *not* deal with precise replication of gesture (Byrne and Russon, 1998) — or even using the same effector on an affordance. For example, Custance *et al.* describe agents that imitate a demonstrator pulling a peg out with fingers by pulling it out with their teeth (Custance et al., 1999).

In summary, we believe that UT is realistic enough to allow the study of human-like imitation, but simple enough to enable us to get at core learning problems relatively quickly. To make explaining the mechanics of the COIL model clearer, we first take a moment to set out the initial task behaviours that we designed to test COIL's aptitude for imitation in this domain. We will then use these example tasks in the description of the COIL algorithm.

5.2.3 The Task Scenarios

UT ships with an environment editor, *UnrealEd* (Digital Extremes, 2000), which we used to create game worlds containing (and, in fact, defined by) the features necessary to complete our initial tasks. These simplified domains are much like the simplified domains used by Roy (1999) and many other developmental-learning researchers. For **Task 1**, the expert demonstrator was a human-controlled bot, and the imitator was the COIL system 'embedded' in an AI-controlled bot. The imitator was programmed to follow the expert, remaining a fixed distance behind and to one side, so as much perceptual information as possible was shared.

The only data made available to the imitator was that received from its own sensors. Latent variables within the expert (such as health) were invisible, just as they are to humans playing the game.

The environment itself was a single cuboid cavern aligned with the world axes[3]. Near each corner, equidistant from the nearest two walls, was placed a **health vial**. Such vials are visible both on screen and on sensors, provided they fall within a bot's field of view (*view cone*). Once picked up they disappear, reappearing (*re-spawning*) after a short fixed time interval. The task demonstrated by the expert was simply to locate and collect vials as quickly as possible.

The expert, imitator and cavern remained unchanged for **Task 2**, but the health vials were replaced by a total of sixteen bots. These bots were recognisably from two different teams; half to be considered 'friends', and half 'enemies'. The task demonstrated was to locate and fire at enemy bots while avoiding friendly ones. A bot which is hit by weapon fire takes damage, affecting its internal health score, and is 'killed' when this score reaches zero, disappearing from sensors. The bots themselves were unarmed and posed no threat to either the expert or the imitator.

A full account of the scope of these tasks to test the breadth of COIL's imitation abilities will be given in Section 5.4. First we describe the detail of COIL's modelling procedures in the following section.

5.3 The COIL System

Our first theoretical COIL model, as detailed below, is as close an adaption as we found possible of CELL to a broader imitation learning context. We now look at each stage in depth: its input, its processes and its output. To aid understanding, we also explain how each stage was applied to Task 1, and compare it with Roy's implementation.

5.3.1 Feature Extraction

The initial input into the first stage of the model is in the form of 'raw data' from the imitator bot's sensors. Each sensor cycle provides data about objects of

[3]So the floor plane was horizontal and the walls were vertical.

note in the environment, specifically the imitator bot's own state, and the visible or audible state (such as location or actions) of other bots, items, weapons, navigation nodes, projectiles, etc. These data are extracted and / or merged into different **channels**. Conversely, a given channel should receive data pertaining to some specific feature of interest in the environment. COIL allows a channel to be one of two types: **A**ction or **P**erception. Henceforth, we will use the term **channel set** to refer to the set containing all channels of a given type.

Our goal for Task 1 was to extract a minimal but sufficient set of features to allow learning of the task. There were only two types of action necessary to complete the task effectively: turning and moving[4]. We thus needed to build biases for detecting these kinds of behaviour into our system (see Section 5.2), and so created two Action channels: one for monitoring rotation in the expert, and one for monitoring motion. Using a piece of memory state in the imitator bot, we designed an algorithm to detect change in attitude of the expert and output the signal to the rotation channel. A similar algorithm detected change in position relative to attitude and fed that data into the motion channel.

Due to the simplicity of the environment, only one type of perceptual information is required for completion of Task 1. We know that the only visible items are the vials to be collected, and that there are no other obstacles, so tracking the bearing of the nearest item to the centre of view is sufficient. This single Perception channel receives data from a more complex algorithm in the imitator bot which uses the item and bot sensors to calculate the relative bearing of each visible item to the expert, and returns the least of these.

Roy's experiment utilised two sensors and a total of three channels. Microphone data were fed into a single Linguistic channel ready for phoneme analysis. Camera data were fed into two Semantic channels: one transformed into colour histograms, and one into shape histograms. Using noisy physical as opposed to clean virtual sensors has disadvantages (see Section 5.2.1), but applying COIL to a real-world problem is nevertheless an interesting open research area (see also Section 8.2). As with Roy's implementation, after feature extraction the relevant data resided in three channels. The next stage is to separate the streams into discrete events which will eventually form the basis for the desired M-E Items.

[4]By *moving* we mean translational movement along the floor plane.

5.3.2 Event Segmentation

Once the relevant data have been diverted into channels, two levels of event detection occur. Top-level **event** boundaries span the given channel set and are determined by a condition on *all* of these channels *simultaneously*. The resulting chunks are known as **A-events** or **P-events** (depending upon the channel set in question). Lower-level **segment** boundaries also span the channel set but, in contrast, are determined by conditions on *each* channel *individually*. In other words, every channel contributes segment boundaries that then span every other channel in the same channel set. **A-subevents** are consequently defined as any continuous sequence of segments within an A-event, across any subset of Action channels. **P-subevents** are analogously defined.

To clarify, let us take a long jump as an example of an A-event. Suppose the A-channels being monitored are running and jumping. Before the athlete starts moving, there will be no activity in either channel. If we suppose the condition for commencing an A-event is passing from 'no activity' to 'some activity', then the start of the athlete's run would trigger the start of an A-event. When the athlete switches from running to jumping, activity in the running channel will cease and activity in the jumping channel will commence. These conditions could be used to define a segment boundary. Finally, when the athlete lands, activity in the jumping channel ceases, leaving both channels dormant. This condition could be used to define the end of the A-event. The A-subevents for this A-event include any subsequence of segments across any subset of the channels, namely (ignoring 'empty' subevents): just the run, just the jump, and the two in sequence.

As an example of a P-event, consider a memory task in which the participant must remember a series of letters held up on cards, as well as the colour and position of those letters. Assume we have three P-channels; one for shape, one for colour, and one for position. When a card is held up, all the channels go from being dormant to being active; a suitable condition for the start of a P-event. In this case, all the channels remain active for as long as the card is visible. When it is hidden, the channel inputs all cease simultaneously; a suitable condition for ending the P-event. No suitable segment boundary conditions arise from this example, although if the letters were to somehow change colour or position then this could provide such conditions. The P-subevents, then, are: letter, colour, position, letter & colour, colour & position, letter & position, and

all three together.

In our experiments, the Feature Extraction process assigned representative 'labels' (real numbers) to pre-defined perceptual classes within each channel. For the rotation channel, the classes were `turning_anticlockwise` (represented by -1), `not_rotating` (0) and `turning_clockwise` (1).

For the motion channel, they were `moving_other_than_forward` (-1), `not_moving` (0) and `moving_forward` (1). The bearing channel classes were `no_items_visible` (100), `item_anticlockwise` (-1), `item_ahead` (0) and `item_clockwise` (1). In addition to the choice of channels, the classes themselves also reflect the imitator's innate knowledge brought to the task at hand and provide a bias toward tractable learning.

Given the data classes, designing event recognisers was relatively straightforward. **A-events** (events which span the Action channels) were triggered by any change in expert bot state. In other words, an A-event started whenever the expert moved or turned (or both) and ceased when it stopped. The gaps between A-events represented times in which the expert was still. A-event segment boundaries were triggered by a change in state in either of the Action channels; a change in the direction of motion from forward to strafing sideways, for example. **P-events** (events which span the Perception channels) were triggered by any change in the bearing channel state. So if, for example, an item that was previously clockwise of the expert becomes ahead, a new P-event commenced. In our implementation, P-events formed a continuous sequence, with the start of a new event triggering the end of the prior. No further segmentation of P-events occurred.

Linguistic-event (L-event) boundaries in Roy's model were triggered by the commencement or cessation of speech input, and so the events themselves consisted of spoken utterances delimited by silence. Segment boundaries coincided with probable phoneme boundaries generated by a Recurrent Neural Network. Semantic-events (S-events) worked slightly differently in that they were not temporal, but a collection of static pictures taken of the object in question from different angles, or **object view sets**. As such, they had no constituent segments, and only contained subevents differentiated by channel span, i.e. colour and shape. This allowed Roy to control the complexity contribution from at least one channel, which we could not.

5.3.3 Co-occurrence Filtering

Co-occurrence filtering is a simple procedure which searches the segmented channel sets for A-events and P-events which overlap in time. Such a pair of co-occurring events is termed an **AP-event**, and shunted to **Short Term Memory** (STM), which is implemented as a queue.

All the channels receive data concurrently due to the imitator's fixed sensor cycle frequency[5]. Since P-events covered the entire time line for the duration of each simulation, every A-event overlapped with at least one P-event[6]. When a coincident pair was found, they were copied to STM as an **AP-event**: a period of continuous action coupled with a period of uniform perception.

Roy's object view sets were timestamped, as were the utterances, which allowed overlap to be established. Consequently, an LS-event in his model consisted of a spoken utterance paired with an object view set.

5.3.4 Recurrence Filtering

The arrival of an AP-event in STM initiates a comparison to be made between the new member and each of the existing members. The constituent A-events of the pair of AP-events under comparison have already been subdivided into segments. Using some predefined metric on A-subevents, d_a, every A-subevent from the new AP-event is compared with every A-subevent from the other. If the distance between them falls below a predetermined threshold, t_a, then the two subevents are marked as matching. The same process is carried out for P-subevents.

Given the matched pairs of subevents generated by the above process, the new AP-event is scanned for co-occurring matches. If a matched A-subevent coincides with a matched P-subevent, their partners in the other AP-event are checked for co-occurrence. If they too coincide, then a recurrent match has been found. Such a match is used to create an **M-E Candidate**, and these are stored in another buffer: **Mid Term Memory** (MTM).

An A-subevent in this instance is a continuous sequence of motion and / or

[5]Again, this may not be as biologically implausible as it sounds; the brain seems to also work to synchronise sensory input (von Stein and Sarnthein, 2000).

[6]In principle, entire P-events could occur in the gap between A-events (i.e. when the bot is stationary), but this didn't happen in our experiments since the environment was largely static and only the bot's action caused changes in its perception.

rotation segments. To measure the distance between such segments, we defined the action distance metric initially as follows:

$$d_a(x, y) = \sum_{c \in C} \frac{|\bar{y}_c - \bar{x}_c|}{|C|} \tag{5.1}$$

where x and y are A-subevents in A-space, C is the set of all channels spanned by x and y, and $\bar{x}_c$ and $\bar{y}_c$ are the mean class values for the given channel c for each respective subevent.

For example, let both x and y be A-subevents representing uniform clockwise turns. So, $C = \{R\}$ (just the rotation channel), $|C| = 1$, $\bar{x}_R = 1$ and $\bar{y}_R = 1$. Then we have:

$$d_a(x, y) = \frac{|1 - 1|}{1} = 0 \tag{5.2}$$

in other words, the two A-subevents are coincident in A-space. If, however, we let y represent a uniform anticlockwise turn (so $\bar{y}_R = -1$), we have:

$$d_a(x, y) = \frac{|1 - -1|}{1} = 2 \tag{5.3}$$

P-subevents are blocks of uniform perception, and the perception distance metric was defined similarly:

$$d_p(x, y) = |\bar{y}_B - \bar{x}_B| \tag{5.4}$$

No sum is necessary, since all P-subevents span and only span the bearing channel (B). The thresholds t_a and t_p were initially both set at 0, so only coincident A- and P-subevents were regarded as matching. Recall that AP-events in STM undergo recurrence filtering whenever a new AP-event arrives. When a pair of matching A-subevents co-occurs with a pair of matching P-subevents, then one of the A-subevents is taken as representative and coupled with one of the P-subevents. This couple is an **M-E Candidate**, which is then added to MTM. Examples of M-E Candidates could include a 'clockwise turn' coupled with an 'object clockwise', and a 'forward motion' coupled with an 'object ahead'.

Bearing in mind that L-subevents in Roy's implementation consisted of sequences of phonemes, the metric d_l that he used was an acoustic distance metric based on the likelihood that two sequences were generated by the same Hidden

Markov Model. He restricted the search by only comparing phoneme sequences of less than one second duration containing at least one vowel. S-subevents were either colour or shape view sets in the form of histograms. The visual distance metric used, d_s, was based upon a χ^2-test for histogram similarity. The match thresholds were set relatively low, and so many Lexical Candidates were created. The metrics Roy selected were well-established; unsurprisingly, there are no conventions for measuring the distance between two generic actions or perceptions. Also, Roy assumed 'close' temporal proximity of similar LS-events (justified by his study of highly repetitious infant-directed speech), and implemented STM as a queue of around just five items. This, in turn, capped the number of comparison operations necessary for each new AP-event. We could not make a similar assumption, as action-perception pairs which arise from a task demonstration could be separated by significant time intervals. Thus our STM buffer was much bigger (typically 25 items), which increased the number of necessary comparisons and affected the efficiency of the process (see also Section 6.1.2).

5.3.5 Mutual Information Filtering

Providing the number of Lexical Candidates in MTM exceeds some fixed minimum, mutual information filtering occurs whenever recurrence filtering generates a new candidate. Before attempting to explain the process, some more terms need to be defined:

- **A-space** and **P-space** are metric spaces with metrics d_a and d_p respectively.
- An M-E Candidate is equivalent to a point in A-space, coupled with a point in P-space. These points are known as the candidate's **A-** and **P-prototypes**.
- An **A-unit** is a sphere in A-space of radius r_a, with an A-prototype at its centre (**P-categories** are defined analogously).
- An A-unit coupled with a P-category is called an **M-E Item**.
- An M-E Candidate **matches** an M-E Item if the candidate's A-prototype falls within the item's A-unit, and the candidate's P-prototype falls within

the item's P-category.

And so, the algorithm runs as follows:

1. An M-E Candidate is selected. Its A-unit and P-category are initialised to have sufficiently small radii so as to contain no other A- or P-prototypes.
2. r_a is increased until another A-prototype falls within the A-unit[7].
3. The mutual information for this configuration of radii is calculated (see Appendix B for method and example). If it exceeds the previous maximum, then it is stored along with the current radii configuration.
4. Steps 2 and 3 are repeated until every other A-prototype has been included in the A-unit.
5. r_p is increased until another P-prototype falls within the P-category. r_a is reset to its initial state.
6. Steps 2 to 5 are repeated until every other P-prototype has been included in the P-category.
7. Steps 1 to 6 are repeated for every M-E Candidate in MTM.
8. If the maximum mutual information exceeds some predefined threshold, then the M-E Candidate and its optimal radii are used to create an M-E Item (see above). This is stored in **Long Term Memory** (LTM). The M-E Candidate, and all those that match the new M-E Unit, are removed from MTM.

For Roy, an L-prototype was a phoneme sequence in L-space, the space of all such sequences. An L-unit was therefore a sphere of phoneme sequences that 'sound sufficiently like' the prototype. An S-prototype was a colour or shape view set in S-space, the space of all such view sets. An S-category was thus a sphere of view sets that 'look sufficiently like' the prototype. So, a Lexical Unit was effectively a spoken word paired with a viewed object, both with allowances

[7]We achieved this by pre-calculating the pairwise distances between all A- and P-prototypes in MTM; a number of calculations quadratic in the number of M-E Candidates.

for individual variation. Roy used a linearly-interpolated prior to smooth mutual information values for infrequently observed pairs, and later versions of COIL also include this function.

5.3.6 Generating Behaviour

Half of the challenge of imitation is acting upon what has been learned, but this function is not incorporated into CELL (although Roy does practically demonstrate his acquired language knowledge in other applications; Roy, 1999, ch. 6). Therefore we have added modules to COIL to complete the 'imitation loop' — to facilitate acting upon acquired behavioural knowledge. These modules, along with their theoretical basis, are the subject of this section.

We have seen how, through observing the completion of a given task by an expert, COIL can construct a 'dictionary' describing the apparent action-perception relationships that are required for, or are a consequence of, the completion of that task. However, the fundamental question of how to act upon said knowledge remains, and is nontrivial. To answer this question, we need to consider what the resultant M-E Items represent in more detail.

In section 5.2, we suggested that *motivation* implies a perception that triggers an action, and *expectation* implies an action that predicts a perception. Recall that an M-E Item is an A-unit coupled with a P-category, and that these, in turn, are derived from an A-subevent which coincides with a P-subevent. The temporal priority of these subevents indicate whether a given M-E Item is in fact a Motivation or Expectation Item. If the initiation of the corresponding P-subevent *preceded* the A-subevent (the perception preceded the action), then it is a Motivation Item; if the A-subevent was initiated first, then it is an Expectation Item.

We discuss the *correspondence problem* (Nehaniv and Dautenhahn, 2002) in relation to UT at length in Section 9.2, but for now we make the simplifying assumption that the expert and imitator have *similar embodiment*; their avatars in the game world have identical body configurations. We can thus write correspondence libraries for the imitator (see Section 3.1.3) which contain simple one-to-one mappings for both perceptions (e.g. `item_ahead` ⇒ `item_ahead`) and actions (e.g. `moving_forward` ⇒ `move_forward()`). This allows the imitator to

search its acquired Motivation Items for those which match its perceptual state (via the P-category), and retrieve candidate actions for execution (via the A-unit). If the Motivation Items cover the imitator's perception space, then the above method provides a complete specification for behaviour, albeit with a simple reactive mapping between state and action. Expectation Items are not as obviously applicable in terms of determining action, and for Tasks 1 and 2 the reactive behaviour provided by Motivation Items alone has been adequate for effective imitation.

Recall that the Perception channel classes for Task 1 related to the possible positions of the nearest item to the centre of the bot's view cone: `no_items_visible`, `item_anticlockwise`, `item_ahead` and `item_clockwise`. Having observed the expert collect vials for some period of time, the imitator switches into acting mode and begins to 'observe' its own perceptual state instead of that of the expert. At the instance at which acting commences, the imitator searches LTM for Motivation Items which match its current state. If there is more than one, the Motivation Item with the highest mutual information value attached to it is chosen. If none match, then the imitator has not learned what to do in this circumstance; depending on the experimenter's preference, it either returns to observing, or is forced to take a random action in the hope of entering a new perceptual state.

Having decided on a Motivation Item to 'act upon', the action that is 'most representative' of its component A-prototype is selected for execution. This is calculated by measuring the distance between the A-prototype and contrived 'pure actions' (that is, a set of A-subevents, $Y = \{y_i\}$, for which each $\bar{y_i}$ is equal to an action class label — see Equation 5.1). The action classes which are comparable depend upon which channels are spanned by the A-prototype. The action closest to the A-prototype in A-space (measured using the action distance metric in Equation 5.1) is then executed.

For example, suppose that the selected Motivation Item contains an A-prototype recorded from the Rotation channel only, where the expert turned clockwise for 90% of the time, then paused for 5%, and finally turned anticlockwise for 5%. The mean rotation class value for this A-prototype is:

$$\bar{x} = (0.05 \times -1) + (0.05 \times 0) + (0.9 \times 1) = 0.85 \tag{5.5}$$

bearing in mind that the rotation class labels (and thus the 'pure actions') are `turning_anticlockwise` (-1), `not_rotating` (0) and `turning_clockwise` (1). So, the closest action is `turning_clockwise`, which has a distance of:

$$d_a(0.85, 1) = |1 - 0.85| = 0.15 \tag{5.6}$$

The action selected for execution is therefore `turn_clockwise()`. The particular action classes that the imitator can recognise in the expert represent a bias in COIL for recognising actions in the imitator's own repertoire. As we mentioned above, this can be partly justified due to the similar embodiment of the imitator to the expert. The command `turn_clockwise(d°)` (i.e. with an angle parameter), for example, exists as a method in the bot controller module.

For Task 1, the imitator was programmed to retrieve a new action every time its perceptual state changed (checked at each sensor cycle), or to repeat the previous action if the state remained unchanged. We defined the actions typically to be short and discrete (such as `turn_clockwise(20°)`), for several reasons:

1. The sensor cycle rate was slow enough that at maximum rotation velocity (for example), whole perceptual classes could be turned through in between cycles, missing an opportunity to change course of action.

2. Most of the available bot commands must be given a numerical parameter, so true continuity of action is difficult (see Section 9.2.2).

3. The game server requires some delay between commands. For example, trying to execute an action every sensor cycle is impossible.

It is quite possible to build apparent continuous motion out of small discrete 'decisions', particularly for embodied agents where their physical plant does a certain amount of its own integration, though it is also possible to build a system with dedicated integration modules (Schaal and Atkeson, 1994; Bryson and McGonigle, 1998; Thórisson, 1999; Bryson, 2005).

Clearly, the quality of the imitated behaviour depends upon both the biases we have built into COIL for the task in question, and the quality of the demonstration given – the next section looks at our results in this area.

5.4 Results

Prior to giving our experimental results, it would seem appropriate to explain our choice of tasks in terms of hypothesising COIL's ability to solve a wider range of imitation problems.

Task 1 is designed to be an elementary task to test the correct functioning of the different system components. Perception space is partitioned, so there are no concurrent competing perceptual classes. However, the observed actions (i.e. turning and rotating) are real valued with varying duration and may overlay multiple perceptual classes: they are not naturally discrete. Also, the bots sense with respect to absolute (world) co-ordinates; COIL must translate these readings into expert-centric co-ordinates to allow easier recognition of the expert's objectives. Task 2 requires all of the above, but additionally COIL must arbitrate attention by prioritising multiply satisfied perceptual classes (see Section 5.4.2 for examples). Task 2 necessitates firing a weapon which, although being more easily discretisable, introduces a new problem: such a short action could get lost or ignored amongst long continuous ones, especially when the sensor sampling only occurs at around 10Hz. We believe that many imitation problems could be constructed from arbitrarily complex combinations of these problems; however, we discuss potential issues with COIL's scalability in the next chapter (Section 6.1.3). We now recap the specific requirements of each task in turn, and give the results of COIL's learning efforts.

5.4.1 Task 1

At the highest level, the behaviour to be imitated in Task 1 was to collect health vials. To achieve this, the imitator monitored the rotation and motion of the expert, and the relative position of the vials in the environment. There were thus two Action channels (*rotation* and *motion*) and one Perception channel (*bearing*). Broadly speaking, actions were delimited by a change of direction (or cessation of motion), and perception by a change of perceptual class. A full specification of Task 1 can be found in the examples of Section 5.3.

As previously mentioned, the Perception space of Task 1 is divided into four mutually exclusive classes: `no_items_visible`, `item_anticlockwise`, `item_ahead` and `item_clockwise`. COIL ideally must acquire sufficient Motivation Items,

Table 5.1: Correct Behaviours for Task 1: **O1** and **O2** represent optimal policies, whereas **NO1** and **NO2** will complete the task, but with wasted rotation. Arrows indicate bearing or direction of motion; × represents *no items visible*.

	Perceptual Class			
Policy	Item ↖	Item ↑	Item ↗	Item ×
O1	Turn ↖	Move ↑	Turn ↗	Turn ↗
O2	Turn ↖	Move ↑	Turn ↗	Turn ↖
NO1	Turn ↗	Move ↑	Turn ↗	Turn ↗
NO2	Turn ↖	Move ↑	Turn ↖	Turn ↖

by observing the expert, to 'cover' this space with the correct actions to complete the task. The actions recognised by COIL were: `turning_clockwise`, `turning_anticlockwise`, `not_turning`, `moving_forward`, `moving_other_than_forward` and `not_moving`. Including the possibility of a null assignment arising from a gap in the observed behaviour, there are therefore seven possible assignments to each perceptual class, giving 28 possible pairings and 16384 possible behaviour **policies**, where a policy is a set of pairs covering all salient perceptual categories[8]. Of these, only four policies will correctly complete the task (see Table 5.1).

Policies **O1** and **O2** are 'optimal' inasmuch as an agent acting accordingly will complete the task in minimal time and with minimal wasted 'energy'. The 'non-optimal' policies **NO1** and **NO2** arise from the fact that turning clockwise $x°$ is equivalent to turning anticlockwise $(360 - x)°$. An agent adopting either of these policies will be able to complete the task, but, unless the imitator and vials have a very specific initial configuration, time and energy is likely to be wasted through surplus rotation. Any other policy will result in an inability to complete the task. An experimental trial consisted of COIL observing, via an imitator bot, the demonstration of the task by a human-controlled bot for 60 seconds. As operators of the expert, we used three different 'tactics' to complete the task:

[8] `moving_other_than_forward` does not translate directly into an executable action. In our experiments, it was mapped to `move_backward()`, but never appeared in any of the learned behaviour. Also, although `not_moving`, `not_turning` and the null assignment all have the same practical outcome for the imitator, as far as COIL is concerned, they are different actions, which is why we make the distinction above.

CW	Tend to turn clockwise if no vials are visible.
ACW	Tend to turn anticlockwise.
Mix	No fixed tendency.

We carried out ten trials for each tactic, for a total of 30 trials. To discover the imitator's learned behaviour, we queried the model directly with the four perceptual classes, rather than by observing the imitator act. Attempting the latter could have lead to further error and subjectivity affecting the results. We used the following program-level task metric (see Section 3.3.1) as the basis for assessing the quality of learned behaviour:

$$d_\pi(s_O, s_E) = \sum_{P_n} d_A(s_O(P_n), s_E(P_n)) \tag{5.7}$$

where s_O is the behaviour learned by the observer on a particular trial, s_E is one of the expected behaviours defined in Table 5.1, and P_n ranges over the four (mutually exclusive) perceptual classes. The metric on actions, d_A, is defined simply as:

$$d_A(a_O, a_E) = \begin{cases} 0 & \text{if} \quad a_O = a_E \\ 1 & \text{if} \quad a_O \neq a_E \end{cases} \tag{5.8}$$

We normalise d_π to give $\bar{d}_\pi$ by dividing by $|\{P_n\}| = 4$ (see Equation 3.20). For each trial, we applied this metric to compare the learned behaviour with each of the four correct behaviours, which allowed us to determine the closest match (s_E where $\bar{d}_\pi$ is minimum) along with a percentage correctness measure:

$$(1 - \bar{d}_\pi) \times 100\% \tag{5.9}$$

Any behaviour which does not score 100% (i.e. which does not exactly match any of the correct behaviours) will fail in the imitation task. The results are shown in Table 5.2.

Note that nearly half of the 30 imitators learned a fully correct behaviour. The majority of the remainder formed only one incorrect association; two formed two incorrect associations (our worst case). Additionally, 27 of the 30 learned policies matched most closely to one of the optimal policies (15 matched O1,

Table 5.2: Results for Task 1: 30 trials were carried out; 10 for each tactic (CW: turn clockwise, ACW: turn anticlockwise, Mix: turn at random).

	% correct behaviour				
Tactic	100	75	50	25	0
CW	5	3	2	0	0
ACW	4	6	0	0	0
Mix	5	5	0	0	0
Total	14	14	2	0	0
	Mean correct behaviour: 85%				

and 11 O2), with only 3 which more closely matched non-optimal policies (2 matched NO1, and 1 NO2). Clearly, COIL performs significantly better than random action allocation would; further discussion and performance comparison is carried out in Chapters 6 and 7. For now we move onto the second task.

5.4.2 Task 2

Until now, Task 2 has only been described in brief (see Section 5.2.3), so the specification that follows adds enough detail to gauge COIL's success, but omits the lowest level technicalities. For this task, the expert aimed to seek and destroy 'enemy' bots, while avoiding 'friendly' bots. The expert was armed, but the target bots were not. This time, the environment was encoded by two Perception channels: *bearing* and *affiliation*. The bearing channel functioned analogously to that in Task 1, replacing items with potential target bots. The affiliation channel received a classification of the nearest bot as either `friend`, `enemy` or `no_target_visible`. The Action channels available to the imitator were *rotation* (identical to Task 1) and *firing*. The firing channel received a binary signal, `firing` or `not_firing`. As per Task 1, A-events were delimited by absences of action, and A-subevents by a class change on either A-channel. P-events were delimited by changes in the affiliation of the nearest target bot, and further segmented into P-subevents by changes of bearing class. The A-space and P-space metrics were defined very similarly to Task 1, with the notable exception

of the firing channel: we defined the distance between `firing` and `not_firing` to be significantly farther than other distances, to reduce the probability that the imitator would confuse the two states.

One of the key differences between the encoding of this task and the previous one is that there are now two Perception channels concurrently receiving data. Therefore, there are three (non-empty) sets of perceptual classes ({`bearing=B`}, {`affiliation=A`} and {`bearing=B`, `affiliation=A`}) concurrently applicable to the expert, as opposed to one ({`bearing=B`}). COIL must learn to correctly prioritise these sets. For example, the imitator might observe the expert firing at an enemy:

$$\{\texttt{bearing=ahead}, \texttt{affiliation=enemy}\} \Rightarrow \texttt{firing}$$

COIL could, however, assign `firing` to just {`affiliation=enemy`}, which would result in the imitator opening fire before the target is in position, or to just {`bearing=ahead`}, which would disastrously result in the imitator firing indiscriminately, at friends and enemies alike. We now describe two ways of measuring success in this task. The first concentrates on the key states:

KS1	{`bearing=ahead`, `affiliation=enemy`}
KS2	{`bearing=ahead`, `affiliation=friend`}

These are key, because correct behaviour in these states is most critical for completing the task. It is worth noting that, although we use the terms *enemy* and *friend*, to the imitator they have no intrinsic meaning; they are just two different types of target. We specify the four possible outcomes relating to these states, in descending order of merit, as follows:

Good	Fire at enemies, avoid friends.
Safe	Avoid both enemies and friends.
Trigger-happy	Fire at both enemies and friends.
Bad	Fire at friends, avoid enemies.

Formally, this amounts to limiting P_n in Equation 5.7 to range only over the key states {**KS1**, **KS2**} – every learned behaviour that covers these states will exactly match (i.e. $d_\pi = 0$) one of the above outcomes. The second way of measuring success is by attempting to define an optimal policy over all of task

Table 5.3: Results for Task 2: 30 trials; 10 for each tactic (as for Task 1). The left-hand side of the table shows the number of imitators exhibiting each type of key state behaviour: **G**ood, **S**afe, **T**rigger-happy or **B**ad. The right-hand side gives the average % correctness score for each tactic.

	Key state behaviours				
Tactic	G	S	T	B	Mean % correct
CW	6	4	0	0	72
ACW	7.5	2	0.5	0	71
Mix	8	1	1	0	65
Total	21.5	7	1.5	0	69

space, and then calculating percentage correctness, using the same process as for Task 1. In this case, we define such a policy as **G**ood in the key states, turn toward a visible enemy, and turn (in either direction) when faced with a friend. Each trial lasted as long as it took for the expert to eliminate all the enemy bots, typically approximately 60 seconds. Tactics **CW**, **ACW** and **Mix** were used analogously to Task 1, again with ten trials each for a total of 30. Results are shown in Table 5.3.

Decimals arise from the fact that, for this task, maximal mutual information was often shared by a number of Motivation Items. Where the number of Motivation Items for two actions were equal, the action assignment for that perceptual class was divided in two (clearly in practise, a method of selecting between these actions would need to be found). Note that over two thirds of the bots tested performed correctly in the key states, and the remainder acquired a definite tendency against shooting friends. The mean behaviour correct score was less for the Mix tactic, because the inconsistency in turning direction provided fewer similar examples for the imitators to form a fully correct policy.

The clear differentiation between firing at friends and firing at enemies shows that COIL had succeeded in prioritising concurrently applicable sets of perceptual classes. Combining this result with that of Task 1, we had some motivation for constructing more complex behaviours to solve hierarchically-structured and / or multi-part tasks. On the other hand, the lack of perfect performance on such seemingly simple problems, and the fact that the algorithm as it stood would

not *improve* performance if it got off to the wrong start in learning were both troubling.

5.4.3 Summary

GTLF and our interest in the characterisation of task learning grew out of a project which focused on imitation learning (Bryson and Wood, 2005). Specifically, our Cross-channel Observation and Imitation Learning (COIL) system (Wood and Bryson, 2007b) is a generalisation to imitation of Deb Roy's Cross-channel Early Lexical Learning system (Roy, 1999; Roy and Pentland, 2002). COIL channels and segments information relating to the perceptions and actions of an agent observed demonstrating a task. It then attempts to find good perception-action bindings by identifying co-occurrence, recurrence and high mutual information. The bindings are used as a specification for imitated behaviour which can then be executed by the learner. The system was implemented and tested in the virtual reality-style computer game *Unreal Tournament* (Digital Extremes, 1999). Agents (called bots) running COIL successfully learned two task behaviours from human demonstration.

In the next chapter, we look at weaknesses in the COIL model, both demonstrated and potential, and show how the use of different representations and algorithms can compensate for them. This discovery ultimately paved the way for GTLF, as described in Part I and implemented in the final chapter of this part.

Chapter 6

From COIL to GTLF

At this point, we felt that COIL had given us some valuable insights into the building blocks we would need for task learning in general, such as a concrete basis for perceptual classes and action elements, an example representation for acquired skills, and so on. We had also gained a working module[1] for learning by observation, along with a deeper understanding of the requirements of this particular method. Nevertheless, we could also see that our strict 'CELL-clone' version of COIL was already hitting obstacles which were likely to prevent it from becoming the powerful, general-purpose tool and model we were aiming for. We examine the most significant of these obstacles in the next section.

6.1 Limitations of COIL

The problems impeding COIL fell into three main categories: issues with the representations inherited from CELL, issues with the algorithms inherited from CELL, and issues with the scalability of our characterisation of task space. We now look at these in turn, and propose some possible solutions.

6.1.1 Issues of Representation

The CELL model seeks to pair a spoken word with a semantic category, where each has associated with it an allowable tolerance for variation. Our direct trans-

[1]Bearing in mind that, at this time, we had not fully envisaged the final modular setup in GTLF.

lation to COIL attempts to pair an action with a perception, again with error boundaries within their respective 'spaces'. A key question is: is this a good model for action-perception linkage or more importantly, given that our goal is imitation, is this a *useful* model?

A Linguistic Unit for Roy was an example of speech which lay in a continuous space of such examples. Two units could be easily compared by an established acoustic distance metric, and part of CELL's aim was to find clusters in this metric space corresponding to words in the English language. In what sense, though, does a generic Action Unit lie in a similar continuous space? For defining reactive behaviour, as is our primary aim, the problem reduces to deciding which action to *initiate* in a given perceptual state. This set of actions is surely *discrete* and finite for a UT bot, and the actions themselves have no established or even easily definable metric relationship to each other. For example, how far is `jump()` from `turn_right()`, and what use would that information be anyway?

The difference in nature between Roy's Semantic and our Perceptual Categories emerges from the way they are formed within their respective models. P-Categories have P-subevents at their core and P-subevent temporal boundaries are defined by innate (pre-programmed) triggers. There is no sensible way to define these triggers *other than* having them respond to changes in perceptual class; classes which the programmer predetermines *may* inform action selection for the given task. Put another way, COIL requires the programmer to *discretise* each perception dimension (channel) into the finest granularity that *could* be necessary to disambiguate perceptual context. This is in fact what happens in Roy's Linguistic channel: the vocal stream is discretised into syllables and pauses, so all words must have boundaries which correspond to syllable beginnings/endings. It is *not*, however, how his Semantic Categories are found. S-Categories created from S-subevents are not temporal as such in Roy's implementation: the association to L-subevents is done manually offline and no triggers need to be defined (see Section 5.3.2). Consequently, no pre-categorisation needs to be done and S-Categories can emerge from clusters formed in well-behaved histogram spaces governed by well-established metrics (Roy, 1999, p. 104).

In summary, COIL assumes discrete action and perception representations, and it is unclear how a well-defined metric could be assigned to such spaces in the general case. This conclusion pointed us toward the more symbolic repre-

sentations we used both in the extension of COIL described below (Section 6.2), and ultimately in GTLF. We were further convinced by results obtained using a symbolic classifier; specifically a decision tree (DT). The classifier took a purely symbolic form of the M-E Candidates that arrived in MTM as input, and output a behaviour specification as described in Section 5.3.6. Run on the same data that was recorded during the experiments of Chapter 5, the DT scored a perfect 100% in Task 1, with non-optimal behaviour learned in only one trial. Interestingly in Task 2, the DT always learned **G**ood behaviour in the key states, but made almost as many mistakes on average as COIL (overall mean 70% correct behaviour).

The final motivation for our change of representation was the potential saving in complexity of the matching and clustering algorithms inherited from CELL, and this is the topic of the next section.

6.1.2 Issues of Process

CELL is designed to emulate early lexical learning, so the task environment is expected to be constrained in certain important ways. One constraint that cannot easily be mapped into the broader domain in which COIL is expected to operate is the **Recurrence Filtering** constraint (see also Section 5.3.4). This limits CELL's 'attention span' to about five consecutive LS-events, and only word-concept pairs which recur within this frame survive the filter. It is, of course, perfectly reasonable to assume high-frequency repetition of keywords, given that the recorded speech is infant-directed. However, our UT imitators cannot always assume equivalent high-frequency repetition of action-perception pairs during the completion of a task: that quite depends upon the task. This begs the question of whether learning UT entirely socially would require 'infant-directed violence'. Notably, some species of predator provide their young with extra practice for the final (and thus, least frequently occurring) stage of a hunt (Caro and Hauser, 1992).

In our first implementation of COIL, we simply slackened the constraint by extending the attention frame to cover a 'large' number of AP-events. In fact, we initially removed it altogether, allowing the filter to compare each new AP-event with *all* of those that had previously been added to STM, but this proved

too computationally inefficient. This led us to consider removing the filter altogether, and replacing it with something more appropriate both to the learning problem and to the new representations mentioned above. Our idea was for short term memory to become some form of **episodic memory** (Tulving, 1983, 2001; Baddeley, 2000, 2001), which eventually became a key part of GTLF (see Section 3.1.3). At first, we reasoned that rather than storing the past five events, we might store an entire task-learning episode. Each game could consist of a sequence of episodes where different tasks are learned and perhaps returned to later. M-E Candidates created from events stored in episodic memory could then be stored in MTM for the duration of a game (or possibly a fixed number of games), and LTM would hold M-E Items generated during all the games of an agent's lifetime.

We then began to envisage episodic memory as a new structure designed specifically to deal with discrete representations and *replace* both STM and MTM. An episode could be represented as a list of observed co-occurring A-subevents and P-subevents, with a count of the number of times that pair recurred with the episode. That way, each AP-event could be processed individually (i.e. no pairwise comparisons would be needed), potentially reducing the computational complexity of the system as a whole. We viewed this as a reasonable match to indexical theories of the hippocampus' role in memory (Teyler and Discenna, 1986; McClelland et al., 1995; Louie and Wilson, 2001), which are best known for capacity reasons: a sparsely encoded representation allows the retention of many events in a finite neural memory. However, they provide the extra (and perhaps more important) attribute of generalised storage of commonly occurring events, which may thus accumulate more 'weight' in the representation. This line of thinking was the basis for the episodic buffer described in Section 3.1.3.

As well as the need for a change to the Recurrence Filtering process, we also identified some potential problems with the Mutual Information Filter (see Section 5.3.5). The algorithm has cubic complexity in the number of elements in MTM (which in general could grow without bound) and exponential in the number of monitored channels (which grows with the complexity of the task domain). Additionally, the probabilities used in the calculation of the mutual information are frequentist (as opposed to Bayesian) approximations, and are consequently very sensitive to noise caused by small frequency counts (i.e. rarely observed

events). Roy tackles this by interpolating these probabilities with priors, but the choice of prior mass parameter required by this technique can have significant effects on the resulting probabilities, particularly if many of the events in question are infrequent. This may well be the case for our applications, so we desired the option of using a more robust method, such as the MLP implemented in our extension to COIL (see Section 6.2).

6.1.3 Issues of Scalability

We have only ever applied COIL to two relatively simple 'local' tasks carried out independently of each other. Even these have highlighted some serious computational issues, which we highlighted in the previous section. However, confronted with the 'full' world of UT, we think it unlikely that even an *ideal* version of COIL could succeeded in learning correct behaviour across a range of tasks. This is mainly due to the complexity of the perception space that would make this possible. We've already established that the number of executable actions is not a major issue, but in a flat COIL architecture, every goal and 'thing worthy of attention' (including memory) in every conceivable in-game task would necessitate its own perception channel.

We conjectured that the solution to this problem was to construct a system which allows for different task-learning frameworks or spaces. To again reference the biological solution, we know that in rats the semantic referent of hippocampal 'place' cells are dependent on the task the subject believes it is engaged in (Kobayashi et al., 1997). Further, the developers of the two dominant cognitive modelling tools, Soar and ACT-R, have found that creating modular 'work spaces' is necessary for replicating human-like learning (Laird and Rosenbloom, 1996; Anderson and Matessa, 1998). Even at an intuitive level, new recruits are generally trained on one aspect of a complex job at a time.

Our initial goal for the extension of COIL was to make improvements at the local level – we needed the core aspects of the tasks to remain constant, both so that we could focus on this goal, and also so that we could make useful comparisons with prior results. Consequently, we put hierarchical task organisation on the back-burner until it came to specifying the full GTLF. Here, it underpins the multi-level roles of perceptual classes, action elements and skills (Section 2.1),

which can be combined with / traded-off against a hierarchical task description (Section 3.5.1).

Before we move on to explaining our first adaptation of COIL, we should state that none of the above critique is aimed at discrediting Roy's application of CELL to the learning problem *for which it was designed*. However, we have now seen some evidence relating to the research questions stated at the outset of the previous chapter, indicating that program-level imitation may not be one of the 'variety of domains' (Roy, 1999, p. 47) in which CELL is readily applicable. Roy makes no such direct claim, of course, and we hope that GTLF, which is a direct 'descendant' of CELL, will prove to be successful in this sphere of learning and others.

6.2 The Next Step

Given that we had achieved some success with COIL, and yet had conjectured a significant number of problems with the system, we were unsure as to its potential both as an adaptive controller and as a learning model. We therefore set out to implement an algorithm which could operate within the COIL framework, while seeking to minimise the impact of the issues we had identified. We specified four desiderata for such an algorithm, based on our analysis up to that point. It should be:

Scalable - both in terms of memory requirements and learning complexity.

Incremental - so that all observations are used and knowledge is consolidated in a natural way.

Rigorous - having output that is interpretable and justifiable through established mathematical argument.

Robust - not prone to failure when processing unusual, unforseen or extreme events.

It would also be preferable for the algorithm to be sufficiently general-purpose to be applicable to other task learning problems. The Bayesian framework seemed like a good place to begin, as it allows each observed event to update prior belief in an efficient and well-defined manner (Bishop, 1995, p. 17). However, there are many algorithms which make use of it, so the question becomes which one to use

in order to obtain the desired posterior beliefs. We chose a multi-layer perceptron (MLP) architecture which, given a certain set of constraints, provides Bayesian posterior probabilities as its outputs. We describe this specific configuration in the next section.

6.2.1 Multi-Layer Perceptron Learning

As explained in Section 6.1.2, the parts of COIL of greatest concern to us were the Recurrence and Mutual Information filters, together with their supporting memory structures. To replace these, we therefore required that the new algorithm receive perception-action data from the Co-occurrence filter and output a behavioural map. In MLP terms, this map looks like a classifier network, which receives perceptual data and assigns it to the appropriate action class. To allow an MLP to learn such a classification, we must translate the observed perception-action pairs into an appropriate set of training examples. Each example should consist of a set of input variables and a set of target variables. In this case, the input variables should correspond to the observed perceptual state of the expert, and the target variables should correspond to the observed action. The question is, what encoding to use for these variable sets?

Following from the discussion in Section 6.1.1, we are assuming now that perceptual classes (such as `item_left` and `no_item_visible`) have no implicit relationship to each other. They do not lie in a metric space and so cannot be represented by ordinal variables. We therefore use a purely categorical *1-of-c* encoding for the input: suppose a given Perception channel has n possible symbolic states. Then symbol i can be represented by a vector of n bits where only the i^{th} bit is set ($1 \leq i \leq n$). If there are m Perception channels then the concatenation of m such vectors produces the complete required binary input vector of length $n_1 + ... + n_m$. If there are k observable actions, then this is equivalent to a classification problem with k target classes, and we can create one output node for each class. We have already stated that it is desirable for these outputs to have a Bayesian interpretation as posterior probabilities of action class membership for a given perceptual state. This is achievable using a softmax activation function for the network output units (Bridle, 1990) and minimising a cross-entropy error function for the network weights (Bishop, 1995, p. 237). After

some empirical testing, we chose to include three hidden units in the network, although the results were not particularly sensitive to this choice. It should also be possible to use Bayesian model selection techniques for selecting the number of hidden units (see Section 6.2.4). Given the above node structure, the network we used was fully connected with a simple feedforward structure, as shown in Figure 6-1.

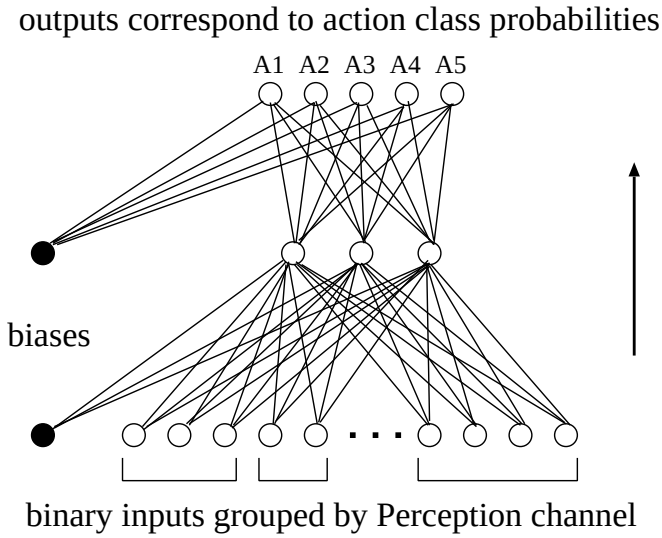

Figure 6-1: Diagram of MLP architecture. The binary input vector is a concatenation of 1-of-c encoded symbols for each Perception channel. There are three hidden units with softmax activation to the outputs, which consequently correspond to posterior probabilities of action class membership. Arrow shows direction of forward propagation; biases are shown explicitly for clarity.

The network training scheme uses Bayesian regularisation with separate hyperparameters for each of four weight groups: first-layer weights, first-layer biases, second-layer weights and second-layer biases (Bishop, 1995, p. 408). Training was by back-propagation for up to 100 cycles of scaled conjugate gradient search (fewer if convergence occurred beforehand), followed by re-estimation of the hyperparameters using the *evidence approximation* technique (MacKay, 1992b). This cycle of re-estimation and re-training was carried out 8 times. The test data for the network consisted of querying all possible combinations of perceptual state, to evaluate the most probable action assigned to that state by the classifier. Finally, these posterior probabilities were marginalised according to the observed data (MacKay, 1992a). Although this last step does not affect the

most probable action class, it can have significant effects if loss matrices are added (see Section 6.2.3 below). Further detail about the MLP structure and training methods we used can be found in Appendix A.2.

Empirical evidence showing the increased *learning* performance of the new algorithm can be found in the next section. Before we examine this however, we review the desiderata set out at the beginning of this section and ask if they are satisfied:

Scalable - as far as learning complexity is concerned, network training time increases only linearly with the number of observed events, as compared to the combinatorics of the original algorithms (see Section 6.1.2). Also, the MLP is a function which requires storage equal to the number of network weights as opposed to a potentially boundless number of stored M-E Items.

Incremental - due to this increase in efficiency and the belief accumulation property of the Bayesian framework, every observation can be taken into account and consolidated with prior knowledge.

Rigorous - the fact that we can interpret the MLP outputs as posterior probabilities is a well-proven property of this type of network and totally independent of the domain in which we're working.

Robust - the parametric re-estimation carried out after each network training cycle serves to minimise any problems caused by local minima relating to, say, weight initialisation.

6.2.2 Results

To evaluate our new algorithm, we tested it against the same data collected for the original COIL experiments.

Task 1

Recall that the first task involved collecting *health vials* from various locations within a game map. During the original experimental runs, the data arriving in channels (i.e. post Feature Extraction) were saved prior to further processing. This allowed us to compare the new learning algorithm on the same data sets. The MLP replaces only the Recurrence and Mutual Information Filtering stages of

COIL, with the first three stages remaining unchanged. For further performance comparison, we also fed this data into a decision tree algorithm, C4.5 (Quinlan, 1992). The results comparing the three techniques are shown in Figure 6-2.

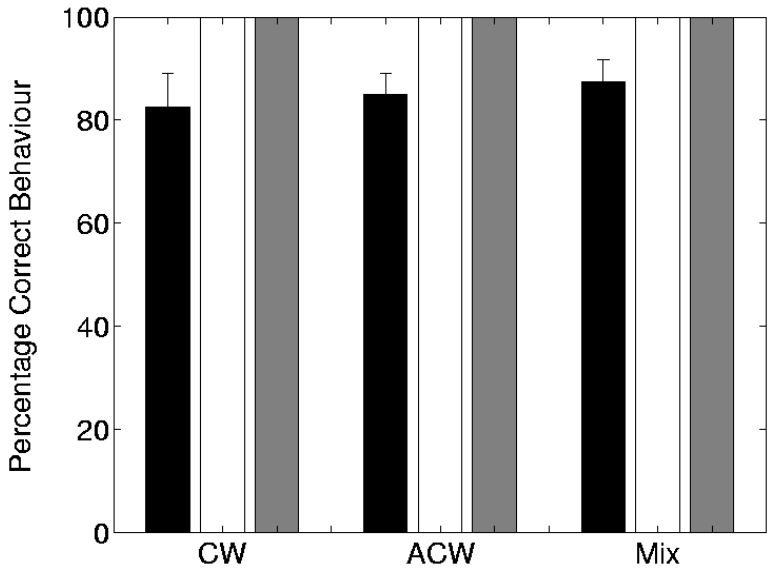

Figure 6-2: Comparative mean performance of the different learning algorithms for Task 1, for each of the different tactics (CW: turn clockwise, ACW: turn anticlockwise, and Mix: turn at random, as described in Section 5.4.1). The black bars correspond to the original COIL processes, the white bars to C4.5 and the grey bars to the new MLP classifier. Error bars show the standard error of the means.

We were able to use the same task performance metric as before (see Equation 5.7), since the trained classifiers are effectively skill functions, and can be queried as such. As can be seen from the figure, the MLP (grey bars) generated universally perfect behaviour for this task, correcting all errors made by COIL's native algorithms (black bars). Interestingly though, C4.5 (white bars) also performs a perfect classification; maybe not too surprising considering the relative simplicity of the perceptual space. The one minor advantage of the network over C4.5 is that the decision tree generated non-optimal behaviour (see Table 5.1) for just one of the trials (the rest were optimal) whereas the network always generated optimal behaviour.

Task 2

The second task required the expert to locate and destroy enemy bots in an environment which also contained an equal number of friendly bots. All algorithms and training methods remained the same for this task as for the previous one. Results are summarised in Figure 6-3.

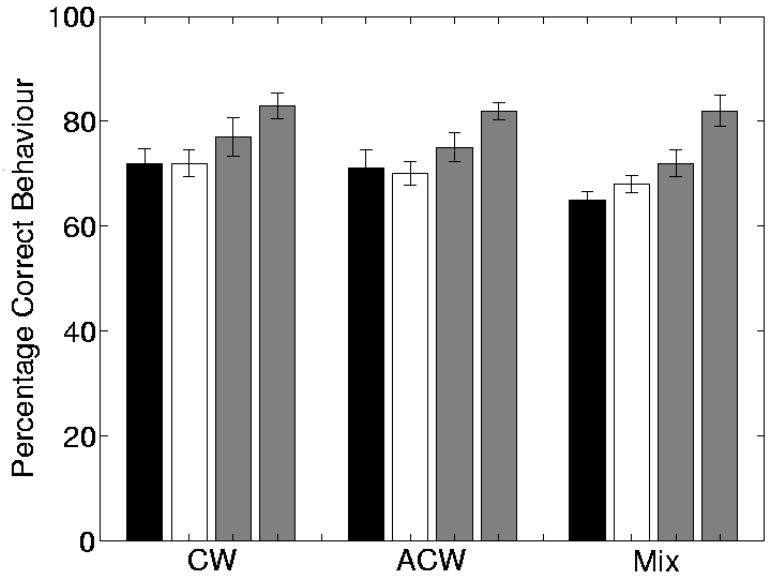

Figure 6-3: Comparative performance results for Task 2; as for Figure 6-2, except that the additional right-hand grey bar for each tactic corresponds to the MLP output moderated by a loss matrix (see Section 6.2.3).

The MLP (left-hand grey bar for each tactic) provides a small but not significant (t-test, $p = 0.05$) increase in performance from both COIL (black bars) and C4.5 (white bars), which for this task performs no better than COIL. Upon inspection of the data, it is clear that for a majority of the trials, the associations that would be necessary to form a fully correct behaviour are never observed. Specifically, most of the misclassifications are made for turning toward an enemy; in the absence of such associations the imitator tended to adopt the dominant turning direction observed and consequently err in either the `enemy_left` or `enemy_right` state. The other common mistake was to fire before the enemy was centred in sights; both were made less by the network than the other algorithms. Performance is further improved, this time significantly (t-test, $p = 0.05$), by introducing a loss matrix to encode prior task knowledge (right-hand grey bar

for each tactic); one of the extentions we go on to talk about in the next section.

6.2.3 Bayesian Extensions

As discussed in Section 6.2.1, the probabilistic interpretation of results possible from the network model is highly desirable. This also allows other Bayesian techniques to be applied to the network and its outputs. We now discuss two such techniques and show preliminary results.

Loss Matrices

In general decision theoretic terms, a *loss matrix* describes the relative penalties associated with misclassifying a given input (Bishop, 1995, p.27). In this case we can describe the matrix as having elements L_{kj} representing the loss resulting from assigning an action A_j to a given perceptual state when in fact A_k should be assigned. Decisions are then made by minimising the *risk*, or equivalently by using the following discriminant to classify a given perceptual state $\mathbf{x}$:

$$\sum_{k=1}^{c} L_{kj} P(A_k|\mathbf{x}) < \sum_{k=1}^{c} L_{ki} P(A_k|\mathbf{x}) \qquad \forall\, i \neq j \tag{6.1}$$

where $P(A_k|\mathbf{x})$ can be obtained from the (marginalised) network output probabilities. The loss matrix values are independent of perceptual state; this is factored in through the conditional probabilities in the discriminant. To demonstrate, we applied a simple loss matrix to the networks generated during Task 2:

$$(L_{kj}) = \begin{pmatrix} 0 & 5 & 5 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \tag{6.2}$$

where A_1 is the `fire()` action, A_2 is `turn_left()` and A_3 is `turn_right()`. This matrix specifies that 'accidentally' firing instead of correctly turning should incur five times greater a penalty than any other misclassification[2]. Informally, one would expect this to be equivalent to giving the imitator the instruction "only shoot if you're sure", prior to acting. The results of applying this matrix

[2]Note a loss matrix with zeros on the main diagonal and ones everywhere else describes a discriminant equivalent to simply choosing the class with the greatest posterior probability.

to the Task 2 network outputs are shown in Figure 6-3 (right-hand grey bar for each tactic). The improvement, as expected, is due to fewer cases of firing before the enemy is in position. Although this is a relatively simple example of the application of this technique, it does demonstrate the ease at which prior knowledge can be formally incorporated into the model, and how it could be systematically altered to test the effect on output behaviour.

Selective Attention

It is likely that for a given task, only a small subset of the full available perceptual state will be required for good performance. So far this subset has been chosen by hand, but the MLP model can enable us to make this selection automatically, within a sound Bayesian framework. This *Automatic Relevance Determination* (Neal, 1996, p. 15) is achieved by using a different hyperprior. Instead of grouping the weights such that there are four independent regularisation hyperparameters, the weights associated with *each input* have their own hyperparameter. These coefficients vary in proportion to the *inverse* posterior variance of the weight distribution for that input. Thus if a given input has a high coefficient value, the weight distribution for that input has a low variance and the input has little bearing on the ultimate classification: the input has *low relevance*. Using a similar training and re-estimation scheme as described earlier, these hyperparameters can be used to determine the relative relevance of the different network inputs, which in this case correspond to different aspects of the environment. Thus we have a method for automatic attention selection within a broader set of channels.

To test this theory, we added a Perception channel to Task 1 which identified the *absolute direction* the imitator was facing, represented by one of four 'compass' symbols. One would expect this set of inputs to have lower relevance than the existing channel relating to the relative bearing of the vials. We carried out a further ten trials under much the same conditions as Task 1. The results are summarised in Figure 6-4.

As expected, the coefficient values for the inputs associated with the new channel are significantly higher on average than the inputs associated with the original channel. The exception to this is the fourth Bearing input which (bearing in mind the 1-of-c encoding) was fully determined by the state of the first three inputs. Given these inputs converged to high relevance, the fourth was

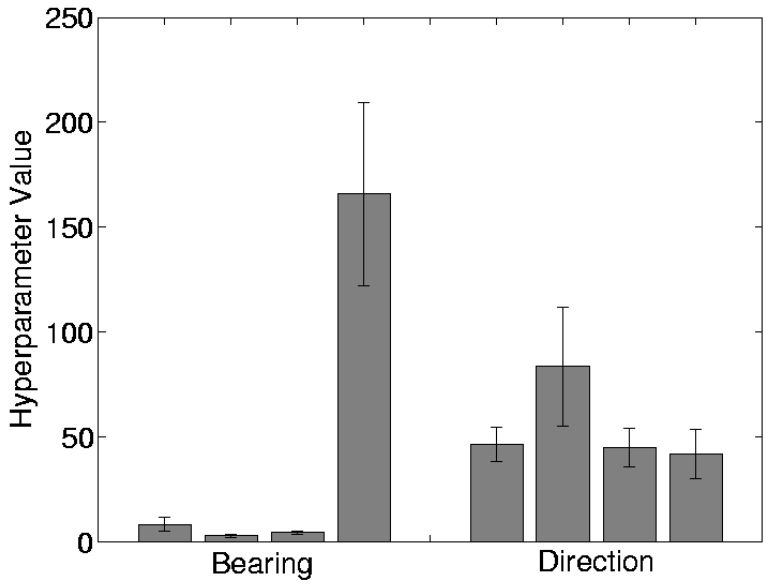

Figure 6-4: Each bar for Bearing represents one of the possible states of the Bearing (Perception) channel: `item_anticlockwise`, `item_ahead`, `item_clockwise` and `no_items_visible` respectively. For the Direction channel, the bars correspond to the states `facing_west`, `facing_north`, `facing_east` and `facing_south` (reference direction is arbitrary). Each input state is associated to a network hyperparameter, the value of which is *inversely proportional* to the relevance of that state for determining course of action.

correctly deemed of very low relevance. Although not demonstrated here, this technique could in principle be used to prune perception space down to make local task learning more accurate and efficient. To allow this, some kind of 'relevance threshold' would have to be chosen, which may well vary from task to task. The method for making such a choice remains an open question.

6.2.4 Summary

Despite COIL's initial success, we identified a number of potential problems looking to future development. Firstly, the representations designed for speech and vision inherited from CELL were not best suited for representing generic actions and percepts. Secondly, some of the algorithms inherited from COIL relied on assumptions which did not hold in the general case, and this adversely affected efficiency. Thirdly, COIL had no facility for representing hierarchical task or behaviour structure, which is likely to have rendered it unusable for complex tasks.

We built an improved version of COIL — "COIL with MLP" — which sought to address the first two of these issues: simpler representations were used, and a generic MLP classifier network replaced some of COIL's native algorithms (Wood and Bryson, 2007a). Experimental results confirmed that the system had been improved, and the immediate success of applying Bayesian techniques to introduce prior knowledge and inform attention revealed some potentially interesting avenues of research. This left us with two obvious options for the next, and what became the final, step of this dissertation project:

1. Follow the success of generalising perception-action representations and algorithms by exploring the idea of a general framework for task learning, which would include the systems we had designed up to that point as special cases.

2. Follow the success of applying Bayesian techniques by fixing our representation / algorithm combination, and conducting more experiments of a similar type with a view to imitating more complex behaviours.

As is clear from Part I, we chose to pursue the former. Although the MLP training algorithm complexity scales well in comparison with COIL, the *global* scalability issues we highlighted in Section 6.1.3 were still of some concern. It seemed a distinct possibility that we would reach a dead-end when trying to imitate more complex behaviours. However, creating a generic framework would allow us to retain the possibility of using MLPs, or indeed any other learning algorithm, while at the same time re-designing the surrounding processes. We could then build in the flexibility we needed to attack more complex behaviour learning. We also conjectured that such a framework would be of more benefit to the field at large, and would open up more interesting research problems (e.g. investigating the role of social learning with respect to individual learning).

We have already described the framework and its basis in detail in Part I, so the subject of the next chapter is our first proof-of-concept implementation of GTLF, including results which indicate its potential as a research tool.

Chapter 7

GTLF Implementation

GTLF, as specified in Chapter 3, is a large and complex modular system. It is likely that most researchers wanting to make use of it will have neither the need, the inclination, nor the time to implement every part. In this chapter, we demonstrate how, using just a few simple modules and algorithms, GTLF can be used to compare firstly the effect of different learning methods, and secondly the difficulty of different observation learning tasks. This should serve both to validate the system, and provide examples for others to follow. For each experiment, we first describe the method used, followed by the GTLF modules we implemented, and then present our results.

Having seen GTLF both in theory and practise, the second half of this chapter reviews a selection of its nearest-neighbour task learning systems in the literature, both robotic and virtual. The relationships between the systems and their relative strengths and weaknesses are discussed.

7.1 Experiment 1: Multi-Modal Learning

To demonstrate the basic workings of GTLF, we chose to design a simple experiment which investigates multi-modal learning. Specifically, we wished to discover the effects that good vs. bad demonstrations during learning by observation has on subsequent trial-and-error learning.

7.1.1 Method

For continuity, we again used Task 1 as described in Section 5.2.3 as our test task; that is, the agent must learn to collect health vials distributed around the environment. This time, however, we created an AI-controlled bot to be the task demonstrator, as opposed to the human-controlled bots of the previous experiments. This allowed us to systematically alter the quality of the demonstration by programming the demonstrator to execute an action element selected uniformly at random from its repertoire with probability r, instead of following the target task policy. An added advantage of using automated demonstrators is that we were able to carry out significantly more trials than had previously been possible[1]. This first experiment was split into three stages. We give an overview here of each stage; more detail can be found in the later **Results** section.

Stage I

The aim of the first stage was to test *if* GTLF was capable of learning the task by observation alone and, if so, *how long* the agents would take to do so. As a control, we used 'perfect' task experts only here — those that always performed according to the target behaviour specification ($r = 0$). We could then use the recorded learning times for this first set of observers to calculate a 'benchmark' trial length; one that is sufficiently long such that a GTLF bot will successfully learn the task with near certainty; and use this in the subsequent stages to test observers learning from less-than-perfect demonstrators. To this end, we ran 30 trials, each one until the task had been successfully learned by the observer.

Stage II

In the second stage we fixed the trial length at the 'benchmark' length calculated from the Stage I results. We then varied the quality of the demonstrations given by varying expert randomness, r (as specified above). The r values used were 0, 0.25, 0.5, 0.75, and 1. An 'expert' operating with $r = 1$ continuously selects actions at random and therefore gives no information about the target behaviour to the learner. This is why we had to use fixed learning times; observers learning

[1]Although note that UT has no quick simulation mode; experiments must still be carried out in real-time, just as for robots.

from such bad demonstrations would never be expected to converge upon the target. Instead, we compared the performance of the learned task behaviour with the target behaviour at the end of the trial, using the same program-level metric as for the COIL experiments. We ran 30 trials for each type of expert.

Stage III

In the final stage of our first experiment we introduced the third variable of interest: learning method. Stage II provided us with results for GTLF bots using learning by observation alone. In Stage III we add trial-and-error learning, for the comparison of three different learning methods:

1. Agents using just observation learning (i.e. Stage II).
2. Agents which start learning by observation, but then switch to learning by trial-and-error half way through the trial.
3. Agents using just trial-and-error learning.

If we revisit the original aim of the experiment; "to discover the effects that good vs. bad demonstrations during learning by observation has on subsequent trial-and-error learning"; then the results of the type 2 agents will be of primary interest. As for the previous stage, we varied r to give five different demonstration qualities and ran 30 trials for each. The results of type 3 agents (also 30 trials) are mainly a frame of reference for the trial-and-error learning of type 2 agents. Before we describe the results of each stage in detail, we first describe how we set up GTLF to cater for this experiment as a whole.

7.1.2 Implementation Specifics

Every agent using GTLF must specify a perception system, (initial) attention strategies, exploratory behaviours, a skill representation, and skill update algorithms. For this experiment, we also needed to implement the observation and trial-and-error learning modules. We chose to assess task performance entirely externally, just as for our previous experiments, therefore we neither implemented Stage 3 (Testing) nor Stage 4 (Reconfiguration). We switched to incremental learning mode (see Section 3.4.1) during trial-and-error exploration to allow for within-episode convergence.

Perceptual System

The perceptual system outputs the perceptual state; the set of perceptual classes occupied by an agent at a given time; given the sensor state. GTLF imposes no particular relationships between perceptual classes, and therefore no particular structure to the perceptual state. However, transferring an idea directly from CELL / COIL (see Section 5.3.1), perceptual classes can be grouped together into *channels* such that for a given instant in time

1. it is impossible for any two classes in a given channel to both apply, and
2. it is *not* impossible for any two classes in different channels to both apply

A natural grouping occurs when each channel corresponds to an *independent feature* of the environment. For example, consider an agent which has a visual sensor capable of discerning `light` and `dark`, and an audio sensor capable of discerning `loud` and `quiet`. According to the constraints above, the only valid allocation is to put `light` and `dark` in one channel, and `loud` and `quiet` in another. The classes `light` and `loud`, say, cannot occupy the same channel, since it is possible for both to apply simultaneously, violating constraint 1. Nor can `light` and `dark` occupy *different* channels, since it is *im*possible for both to apply simultaneously, which would violate constraint 2.

With this system, at most one perceptual class from each channel can apply at any given time. With n perception channels the perceptual state can be represented as an n-dimensional vector $P = (p_1, p_2, ..., p_n)$ with each element p_i representing the perceptual class contributed by channel i. If at a given instant no perceptual classes in channel i apply, then for completeness we say this channel occupies its *null class*, and represent this by $\emptyset_i$. Organising perception into channels in this way can improve theoretical efficiency as task complexity increases (for more details see Section A.3).

For this experiment we define 8 perception channels. The first channel, *vial bearing*, contains the only perceptual classes necessary for *executing* correct task behaviour. The remaining seven contain classes necessary for *learning* the task by observation and trial-and-error:

1. The *vial bearing* channel — monitors the closest vial to the *learner*'s centre

of view. Contains `no_vials_visible`, `vial_clockwise`, `vial_anticlockwise` and `vial_ahead`.

2. The *rotation* channel — monitors the rotation of the learner. Contains `turning_clockwise`, `turning_anticlockwise` and `not_turning`.

3. The *motion* channel — monitors the motion of the learner. Contains `moving_forward`, `not_moving` and `moving_other_than_forward`.

4. The *action history* channel — monitors (i.e. remembers) the previous action initiated by the learner. Contains `turned_clockwise`, `turned_anticlockwise` and `moved_forward`.

5. The *reward* channel — monitors rewards administered by the environment. Contains `reward_received` and `punishment_received`.

6. The *expert bearing* channel — monitors the closest vial to the *expert*'s centre of view. Contains `no_vials_visible`[2], `vial_clockwise`, `vial_anticlockwise` and `vial_ahead`.

7. The *expert rotation* channel — monitors the rotation of the expert. Contains `turning_clockwise`, `turning_anticlockwise` and `not_turning`.

8. The *expert motion* channel — monitors the motion of the expert. Contains `moving_forward`, `not_moving` and `moving_other_than_forward`.

Note that channels 6, 7, and 8 correspond to the Perception and Action Channels respectively as defined for the COIL experiments — these classes will be used by the observation learning module. The remaining channels monitor the state of the learner, and will be used by the trial-and-error learning module. Note that the perceptual class `reward_received` appears in the perceptual state when the learner picks up a health vial, and `punishment_received` appears after the learner bumps into a wall.

In summary, the learner's perception system will map its sensor state to a perceptual state vector of length seven. Henceforth p_i will represent the perceptual class contributed to the state by channel i.

[2]The duplicated names for perceptual classes are for the sake of brevity; i.e. to avoid cumbersome names like `no_vials_visible_to_expert`. This should not cause a problem, since the classes can be disambiguated by reference to their containing channel.

Attention Strategies and Exploratory Behaviours

In these experiments, where different learning methods are used the always occur in sequence rather than in parallel. This allows us to define two pairs of attention strategies; one for observation learning, and one for trial-and-error; making for a total of four (see Section 3.1.2):

- a_E^O — selects the classes necessary for executing exploratory behaviour for observation learning.
- a_L^O — selects the classes necessary for the observation learning module.
- a_E^T — selects the classes necessary for executing exploratory behaviour for trial-and-error learning.
- a_L^T — selects the classes necessary for the trial-and-error learning module.

For observation learning, we wish the exploratory behaviour to move the observer to a good vantage point at a fixed distance and orientation to the expert. The location of this vantage point is calculated automatically upon seeing the expert, within one of the learner's sensor modules, and so can be referenced deictically by the action element `move_to(viewpoint)`. If the expert is not visible, we wish the agent to `turn_anticlockwise()` and look for it. The expert is visible iff $p_6 \neq \emptyset_6$, $p_7 \neq \emptyset_7$ and $p_8 \neq \emptyset_8$, so selecting any one of these classes is sufficient for determining the correct exploratory action to execute. In our experiments, a_E^O selects p_6.

The classes needed by the observation learning module are those relating to the state of the expert; p_6 (bearing), p_7 (rotation) and p_8 (motion); so a_L^O selects these classes (see below for details).

For trial-and-error learning, we wish the exploratory behaviour to mostly correspond to the learner's current approximation of the task behaviour (bearing in mind that this gets updated after every observation in incremental learning mode). Only the bearing of the learner, p_1, is needed for this. However, we also wish the learner to periodically execute a random exploratory action (`turn_clockwise()`, `turn_anticlockwise()` or `move_forward()`), which is handled by the action element `random()`. This can be achieved by referencing the *exploration rate* variable, e, stored in the trial-and-error learning module (see

below). With probability e, then, the learner executes `random()`, and this does not depend in any way on the perceptual state. To summarise, executing a task action requires p_1, and executing a random action has no requirements, therefore a_E^T selects p_1.

The classes needed by the trial-and-error learning module are those relating to the current state of the learner; p_1 (bearing), p_2 (rotation) and p_3 (motion); the previous action of the learner; p_4; and the reward channel; p_5. Therefore, a_L^T selects these classes (see below for details).

Skill Representation

In this implementation, we represent skills as a 2-dimensional matrix U of real-valued utilities. The utility of executing action element a in perceptual state P is given by the value stored at matrix position $U(P, a)$. The behaviour policy itself is then determined by simply selecting the highest utility action element for a given perceptual state.

Observation Learning Module

Compared to COIL and the MLP used in Chapter 6, the observation learning module in this implementation is very much simpler. The basic premise remains the same however; that an action which is observed to be initiated in a given state should be more strongly associated with that state.

The module receives input from the attention strategy a_L^O (see above) those perceptual classes pertaining to the state of the expert; namely p_6 (bearing), p_7 (rotation) and p_8 (motion). It has a one-step memory containing the values of these variables at the previous time step: p_6^{old}, p_7^{old} and p_8^{old}. If the expert's rotation has changed; $p_7 \neq p_7^{old}$; this implies that the expert's rotation p_7 was initiated in state p_6^{old}. By our learning principle, the association between p_7 and p_6^{old} should be strengthened. However, p_7 and p_6^{old} are perceptual states relating to the expert (i.e. allocentric) and, as they are, cannot be used for skill update.

This is where the correspondence library comes in. The module searches its perceptual correspondences for a match to the allocentric class p_6^{old}. The perceptual correspondences for this task are trivial:

Perceptual Correspondences			
Allocentric Class	Channel	Egocentric class	Channel
`no_vials_visible`	6	`no_vials_visible`	1
`vial_clockwise`	6	`vial_clockwise`	1
`vial_anticlockwise`	6	`vial_anticlockwise`	1
`vial_ahead`	6	`vial_ahead`	1

We refer to the matching egocentric class as p_1^{match}. The module then searches its action correspondences for a match to the allocentric class p_7. The action correspondences for rotation are:

Action Correspondences (Rotation)		
Allocentric Class	Channel	Action Element
`turning clockwise`	7	`turn_clockwise()`
`turning anticlockwise`	7	`turn_anticlockwise()`

We refer to the matching action element as a. The module is now ready to output an association datum to the episodic buffer (see Section 3.1.3):

$$(\{p_1^{match}\}, a, 1)_a$$

The data in the buffer at the end of an observation learning episode is used for skill update by the corresponding algorithm (see below). It should be noted that the above process applies in an identical manner for a change in expert motion; $p_8 \neq p_8^{old}$.

Trial-and-error Learning Module

Given the real-time nature of the task environment, together with our 'somewhat semi-Markovian' characterisation of the task (see Appendix A.4), any standard discrete-time Markov-based RL algorithm would have been inappropriate for inclusion in this module. We therefore decided to use a model-free RL algorithm called SMART (Semi-Markov Average Reward Technique), constructed by Das et al. (1999). As well as its superior suitability, it has the added advantage (as its name suggests) of using rewards averaged over time (see below). This has the effect of encouraging our learning agents to collect rewards (i.e. vials) as quickly as possible; desirable in timed trials such as those we are carrying out.

The module receives input from the attention strategy a_L^T (see above) those perceptual classes pertaining to the current state of the learner; p_1 (bearing), p_2 (rotation) and p_3 (motion); the previous action of the learner; p_4; and the reward channel; p_5. The module also contains three pieces of state; p_2^{old} and p_3^{old}, respectively the rotation and motion of the learner at the previous time step; and p_1^{old}, the state of the learner at the previous *iteration of the algorithm*. An iteration is triggered whenever the learner changes state; $p_1 \neq p_1^{old}$; the bot stops; $p_2 = \texttt{not\_turning}$ and $p_3 = \texttt{not\_moving}$; or the bot receives a reward; $p_5 \neq \emptyset_5$.

As we noted in Section 3.1.3, we assume that the trial-and-error learning module contains a 'self-correspondence library', much like the action correspondence library in the observation learning module, that in this case can be used to retrieve the agent's previously executed action element, a, given its perception (i.e. memory) of that action, p_4. For example, a memory stored in channel 4 that the agent `moved_forward`, corresponds to the element `move_forward()` in the agent's action repertoire.

For each iteration, the module generates a reward signal r as follows:

$$r = \begin{cases} 10 & \text{if} \quad p_5 = \texttt{reward\_received} \\ -1 & \text{if} \quad p_5 = \texttt{punishment\_received} \\ 0 & \text{if} \quad p_5 = \emptyset_5 \end{cases} \tag{7.1}$$

The module keeps track of five further variables to allow for average reward RL: e, the exploration rate (see above); α, the learning rate; τ, the elapsed time since the last iteration; t, the total time of the learning episode so far; and c, the total reward gained in the learning episode so far. The average reward, g, at a given iteration is therefore given by $\frac{c}{t}$. Both the learning rate, α, and the exploration rate, e, decay slowly to 0 according to a Darken-Chang-Moody search-then-converge process (see Appendix A.4.1; Darken et al., 1992; Das et al., 1999, for details). Access to the current utility values for the skill being learned, U_{old}, is also required.

Given that all the necessary variable values are available, the following update rule is used to find the new utility value:

$$U_{new}(\{p_1^{old}\}, a) = (1-\alpha)U_{old}(\{p_1^{old}\}, a) + \alpha\left(r - g\tau + \max_b[U_{old}(\{p_1\}, b)]\right) \tag{7.2}$$

In words, the new utility value ($U_{new}(\{p_1^{old}\}, a)$) is a linear interpolation (where the relative weighting is determined by the learning rate, α) of the previous value ($U_{old}(\{p_1^{old}\}, a)$) with the sum of the reward just gained (r) and the maximum utility value for any possible action in the previous state ($\max_b[U_{old}(\{p_1\}, b)]$) *less* the reward expected during the time interval ($g\tau$). So, if the reward just gained is less than the reward expected for that time interval, $r - g\tau$ contributes negatively to the update, and vice versa.

The new utility value is output as an association datum to the episodic buffer:

$$(\{p_1^{old}\}, a, U_{new}(\{p_1^{old}\}, a))_a$$

Since we have set GTLF to run in incremental mode during trial-and-error learning, this datum is processed (i.e. the skill is updated) immediately and then removed from the buffer readying it for the next iteration.

Skill Update

Skill update occurs at the end of an observation learning episode, and after each iteration for trial-and-error learning. We look at each case in turn.

For observation learning, recall that the association data have the form:

$$(\{p_1^{match}\}, a, 1)_a$$

We use a very simple learning rule adapted from Bryson (2001, p. 153) to update the utilities in the skill matrix. For each datum in the episodic buffer:

$$U'(\{p_1^{match}\}, a) = U_{old}(\{p_1^{match}\}, a) + \delta \tag{7.3}$$

where δ is a free parameter, with a value of 0.1 in our experiments. Once all the data has been processed, each row of the matrix is renormalised:

$$U_{new}(\{p_1^{match}\}, a) = \frac{U'(\{p_1^{match}\}, a)}{\sum_b U'(\{p_1^{match}\}, b)} \tag{7.4}$$

For trial-and-error learning, skill update occurred after each new datum. These data have the form:

$$(\{p_1^{old}\}, a, u)_a$$

Where u is the new utility value calculated in the trial-and-error learning module. The update, then, is trivial:

$$U_{new}(\{p_1^{old}\}, a) = u \tag{7.5}$$

since all the work has already been done. This illustrates the trade-off between the amount of work done within the learning modules and the amount done during skill update. There are likely to be a number of different ways the workload could in theory be distributed, and which is chosen depends on the constraints of the problem in question. For example, if skill update occurs offline between episodes, it makes more sense to do resource-intensive processing at this stage, as opposed to within the learning modules at each time step.

7.1.3 Results

We now give our results for each stage in turn (since each stage builds upon the results of the previous), along with any implications that we have drawn. Where significance is claimed and a p value is given, we have used a two-tailed t-test.

Stage I

Before we could measure learning time for each trial, we had to define reasonable stopping criteria which indicated when convergence had occurred[3]. We therefore stipulated that a trial ended when all of the following were satisfied:

- A *complete* behaviour had been learned (i.e. one that defines an action in every possible perceptual state).
- The behaviour mapping had not changed for some predetermined length of time.
- The difference in utility values between successive skill updates (which could be seen as the 'learning error') had fallen below a predetermined threshold.

[3]For observation learning, there are no formal guarantees of convergence. Regardless of the target behaviour, if the demonstration changes or otherwise fails to provide sufficient information, then learning may never converge and in fact may begin to diverge.

Across 30 trials, then, the mean time taken by agents learning by observing perfect experts to converge, in accordance with the above definition, was 106.6 seconds, with a standard deviation of 48.7 seconds.

We had then to convert this result into a 'benchmark' trial time for use in the subsequent stages, which we achieved by defining an appropriate **tolerance interval** (Croarkin and Tobias, 2003). In brief, a tolerance interval for a given measured quantity (in this case learning time) estimates the range of measurements that will with probability p contain a pre-specified proportion q of the population. We wished to define a tolerance interval for learning time that with 99% probability ($p = 0.99$) contained 99% of GTLF bots ($q = 0.99$). Finding the tolerance factor for a population sample of size 30 using the method described in Appendix B, we calculated the upper bound of this tolerance interval to equal 288.5 seconds. A benchmark trial length of 300 seconds (i.e. 5 minutes), then, should be adequate to all but guarantee convergence for agents observing perfect experts. We used this length of trial moving into Stage II: learning by observation for variable qualities of demonstration.

Stage II

The results from Stage II are shown in Figure 7-1. Both the target behaviour and the program-level metric used to calculate percentage match were the same as those used for COIL Task 1 (see Section 5.4.1). As predicted from Stage I, the observers learning from perfect demonstrations unanimously scored a 100% match to the target behaviour. Each drop in expert performance caused a highly significant ($p < 0.01$) drop in learner match percentage, also as would be expected. One notable feature of the results is that the degradation appears to be somewhat nonlinear — the drop between 50% and 75% expert randomness is greater than previous drops. This could be because at 75% (and 100%) randomness, the expert began sometimes to get stuck in certain areas of the task environment (e.g. in a corner), thus providing the observer with no more clues about the target behaviour for the remainder of the trial. It is also not clear why 100% random demonstrations would *consistently* result in a 25% match to the target behaviour. We believe that this is probably an anomaly of our results set, and that in a larger sample there would a few who chance upon a 50% match, and equally a few who end up with a 0% match. Given these results for observation

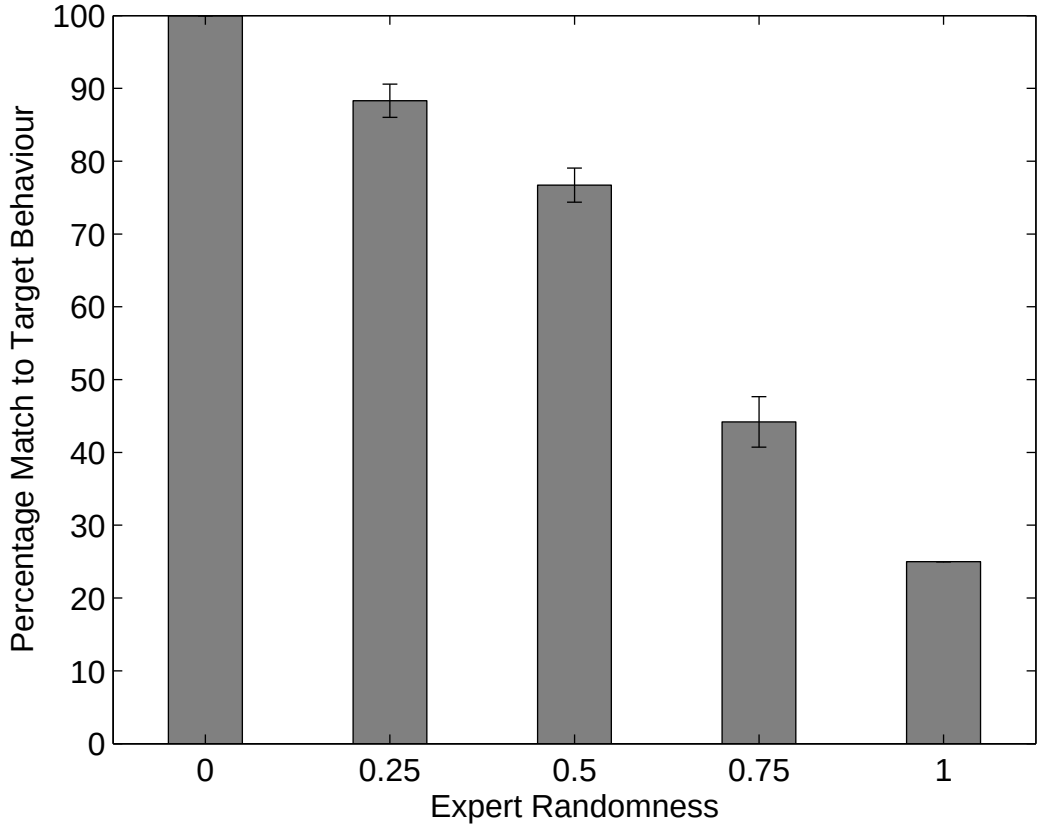

Figure 7-1: Comparative performance of agents learning by observation as expert performance degrades. Error bars show standard error of the mean.

learning alone, we now move onto results comparing different learning methods.

Stage III

The results from Stage III are shown in Figure 7-2. The results from Stage II are also included (the dark-grey bars) for ease of comparison. The first thing to note is that, for all but the perfect demonstrations, trial-and-error learning does compensate for poor expert performance by improving the resulting match to the target behaviour. For the particularly bad demonstrations (75% and 100% randomness), this improvement is highly significant ($p < 0.001$). The shallower decline of 50-50 learning can be explained by the fact that areas of task space which were not visited by bad experts can be explored during the trial-and-error phase.

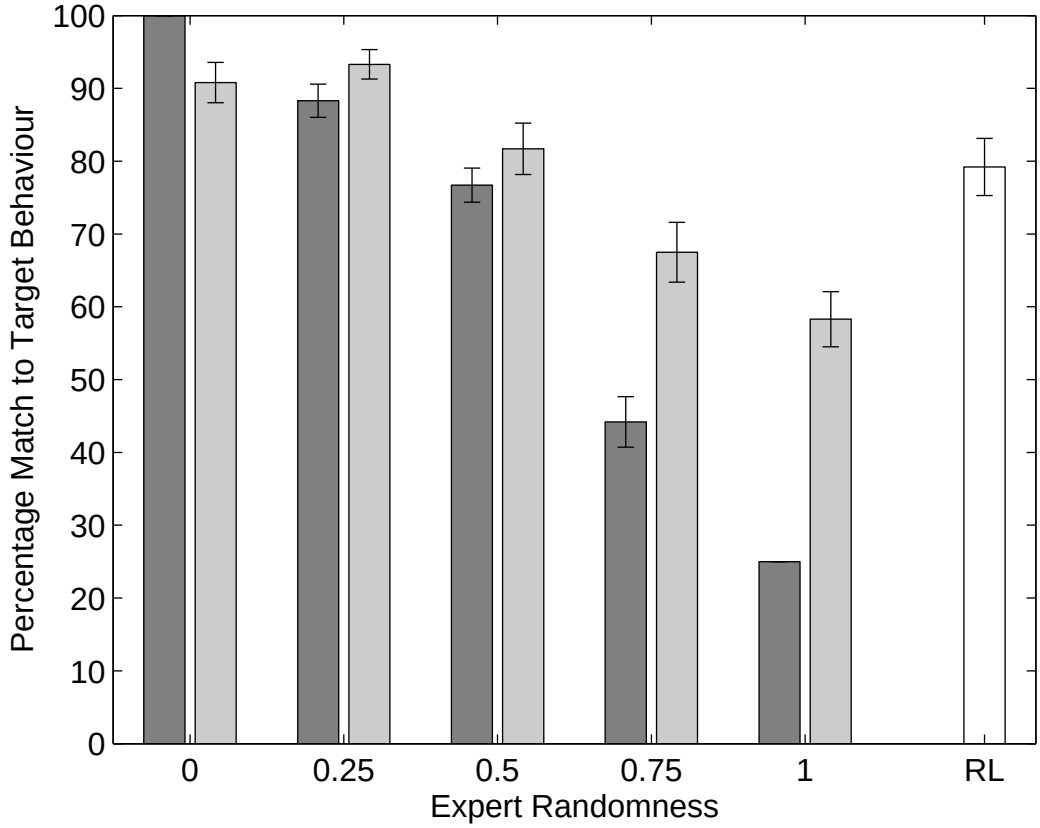

Figure 7-2: Comparative performance utilising different learning method ratios as expert performance degrades. The dark-grey bars represent pure learning by observation (Stage II). The light-grey bars represent a 50-50 split, first observation then trial-and-error. The white bar on the far right represents pure trial-and-error. Error bars show standard error of the mean.

We also note that for the better demonstrations (0% and 25% randomness), the mixed learning method performs significantly better than pure RL ($p < 0.05$), indicating that some useful behavioural information from the observation stage is being retained and built upon. Perhaps the most interesting result for the mixed learning agents is at 25% expert randomness, where this approach both significantly outperforms pure RL ($p < 0.01$) and pure observation ($p < 0.1$). It may be interesting to perform further tests at a finer granularity in this region; i.e. for expert randomness between 20% and 30% at 2% intervals; to see if there is an apparent optimum for using the mixed learning approach. It may also be interesting to vary the proportion of learning time spent using each method; i.e.

25% observation followed by 75% trial-and-error, and vice versa.

Metrics and Performance in Practise

Before we move onto the next experiment, we should briefly justify our universal use of a program-level metric for measuring task performance. Firstly, it should be borne in mind that none of the agents, whether learning by observation or trial-and-error, have knowledge of the metric by which their performance will end up being measured. The observers are slaves to their learning algorithm; blindly replicating whatever behaviour is demonstrated to them. They have no knowledge of whether this behaviour is 'good' or 'bad', and no recognition of the existence of a task or its goals — they simply imitate. In exactly the same way, the trial-and-error learners are slaves to their internal reward function; they act to maximise long-term reward, but effectively have no explicit knowledge of performing a task. Ultimately, then, for both these types of agents, success in this task depends upon the agent designer's choice of algorithm. If the algorithm provides sound learning advice, performance will be good; if not, it will be bad. This should highlight the designer's involvement in any experiment of this type: the results and conclusions drawn can never be truly objective.

However, we do not believe this is necessarily a problem, as long as it is acknowledged. As we have argued previously, a task must always be defined by some agent, in this case the agent designer himself, and cannot be independently or absolutely defined. Therefore, performance can only ever be measured in a subjective sense, and so long as the extent of that subjectivity is understood, then the conclusions drawn can be interpreted in the light of this. Our choice of program-level metric is exactly that: our choice to define the task itself in terms of that metric. The metric and the task are the same thing. We have also designed the internal algorithms of the learning agents as best we can to guide the agents towards good performance in terms of this metric. So, the influence of the agent designer in this task learning problem is admittedly ubiquitous, but we suggest that this is likely to be transparently the case in many similar research scenarios.

7.2 Experiment 2: Different Target Behaviours

The second experiment we carried out was somewhat simpler than the first; to compare time to convergence for a number of different target behaviours when learning by observation. This should serve to give a better idea of the relative difficulty of learning problems when defined in terms of GTLF constructs.

7.2.1 Method

For this experiment, we slightly adapted the task environment from the previous experiment to add an extra dimension of complexity. As before, the task to be completed was picking up vials. This time, the cavern also contained two groups of bots: one group designated as *enemies* and one as *friends* (much the same as for COIL Task 2, Section 5.4.2). The vials on one side of the room were 'unguarded' (i.e. they had no bots near them). For the vials on the other side of the room, one was guarded by enemy bots, and one was guarded by friendly bots. Given this setup, we were then able to define the following target behaviour variants to compare:

1. Pick up all vials, regardless of whether or not they are guarded.
2. Pick up only guarded vials.
3. Pick up only vials guarded by friends.

Behaviour 1 is superficially the same target behaviour as for the previous experiment. The difference is that the learning agent must explicitly learn that the bots in the environment should not have any bearing on its vial-collecting behaviour. In advance of running the trials, we predicted that these behaviours are listed in ascending order of learning difficulty, and therefore should also be in ascending order of mean learning time. We used AI-controlled demonstrators as before (all operating with 0% randomness), and ran 30 trials of observation learning for each behaviour, where each trial ran to convergence (as defined above).

7.2.2 Implementation Specifics

As far as GTLF implementation is concerned, this remained in most respects identical to that used for observation learning in the previous experiment. We

added one more perception channel for monitoring the bot closest to the learner's centre of view, which contained the classes `no_bots_visible`, `enemy_bot_ahead`, and `friendly_bot_ahead`.

7.2.3 Results

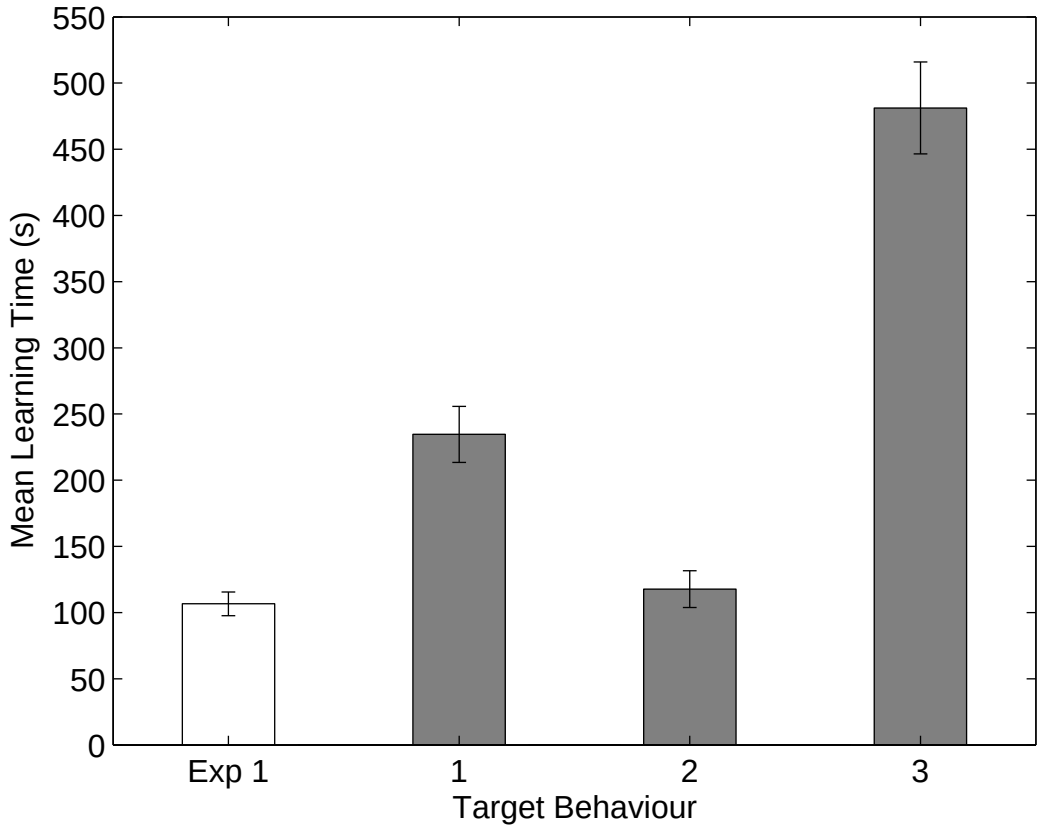

Figure 7-3: Comparative learning (by observation) times for different target behaviours. White bar shows target behaviour from Experiment 1 for comparison (pick up vials, no bots in environment). Target behaviour 1 was to pick up vials regardless of other bots. Target behaviour 2 was to pick up only guarded vials. Target behaviour 3 was to pick up only vials guarded by friendly bots. Error bars show standard error of the mean.

The results from Experiment 2 are shown in Figure 7-3. The target behaviour from Experiment 1 is also included (white bar) for ease of comparison. The first thing to note is that our prior intuition was incorrect. Target behaviour 2, where

only guarded vials should be collected, is apparently easier to learn than target behaviour 1, where any vials can be collected. Although this seems surprising at first, we offer the following explanation as to the cause. All of the new target behaviours require the observer to learn the significance of the other bots in the environment. This is why behaviour 1 takes more than twice as long to learn as the target behaviour from Experiment 1, even though they are identically expressed in the environment. However, for target behaviour 2, the demonstrator's actions are focused around the other bots, since it only ever collects vials in their vicinity. This gives the observer twice as many opportunities to discern the relevance of these bots when compared with behaviour 1, which visits guarded vials only half as often. In other words, demonstrators executing behaviour 2 provide information much more efficiently than those executing behaviour 1, and therefore allow learning to converge more quickly. In fact, the bottleneck for behaviour 2 seems to be learning what to do with the vials, since the mean learning time is only marginally (i.e. not significantly) longer than learning this task without any bots being present (Experiment 1). Target behaviour 3 introduces the requirement that the learner must distinguish enemy bots from friendly bots, and coupled with the fact that 'behaviour samples' involving enemy bots are rarely demonstrated, this behaviour follows our prediction of being much harder to learn.

We conclude the first half of the chapter by observing that even simple experiments such as these have prompted some interesting discussion. It is essentially this fact even more than the results themselves which validates GTLF as a potentially valuable research tool. In the next section we seek to augment this validation by attempting to determine where GTLF sits in the literature, and thus how it can be best put to use from now on.

7.3 Existing Task Learning Systems

Back in our introduction to the subject, we considered some biological examples of task learning, to examine and draw inspiration from those methods already in place in nature (Section 2.3). Now our framework has been presented, we look at existing artificial task learning systems, to see where ours can explain, extend or improve the state-of-the-art, and vice versa. We focus on systems which fulfil

the following criteria to the greatest extent:

1. The system theory bears a resemblance to ours inasmuch as it is a framework for co-ordinating learning of multiple kinds at multiple levels.

2. The system implementation domain bears a resemblance to ours inasmuch as it involves virtual agents learning complex tasks in a real-time setting.

3. The system is the subject of a recent publication.

Artificial systems, by definition, will fall into two sub-categories, which we will label *robotic* and *virtual*. The former pertains to any system which has been designed for and / or tested using man-made material agents; the latter applies to non-material agents. It so happens that robot task learning systems have, until now, better fulfilled the first criteria, whereas (for obvious reasons) virtual task learning systems better fulfil the second.

7.3.1 Robot Task Learning

Reinforcement learning in robots is inherently difficult due to the generally large number of trials needed, necessity of real-time learning, time and expense in setting up experiments, and so forth. For these reasons, many robot task learning systems concentrate on social learning methods such as imitation and instruction[4]. We now review two of the most relevant:

Nicolescu and Matarič

Nicolescu and Matarić (2001, 2002, 2003, 2007) present a system for task learning by imitation motivated by two related problems: the need to design robots capable of natural interaction with humans, and the desire for robots to be able to expand their capabilities through learning. The latter constitutes the more significant overlap with our work.

The learning robots are assumed to initially possess a library of behaviours (in the BBAI sense) stored using *action-based representations*. This is the first of the main contributions, and divides behaviours into two parts; *abstract* and *primitive*. The abstract part of behaviour relates to perception, and defines the

[4]Programming could be included here, or simply regarded as prior knowledge.

pre-conditions necessary to initiate a behaviour and the post-conditions which indicate when the goal of the behaviour has been satisfied. The primitive part relates to actions that can be executed by a given robot in a given task domain, and so the linked pair provides the coupling of perception and action. By using representation to abstract the 'logical' part of behaviours from the active part, it is possible for learned tasks to be transferable between specific task-robot scenarios. This enhances standard Behaviour-Based Systems, in which perception and action have no behaviour-independent representation, and cannot therefore be de-coupled, generalised or subjected to higher-level reasoning processes.

In practise, the robots learn by following human demonstrators around the task environment, responding to instructions given at key points. As the demonstration is experienced, the robot monitors its behaviour repertoire, observing when the pre- and post-conditions of the various abstract behaviours are met. This firing sequence defines one possible path through the task, a *behaviour network*, which can then be executed by the robot. On a test trial, the robot similarly monitors its abstract behaviours, but instead of following it executes the associated primitive behaviours which should carry the robot through the task. This learning method allows the robot to acquire an approximation of the task behaviour in just one demonstration. However, mistakes can be corrected by repeated observations or by corrected trials. In the latter case, the robot carries out the task according to the behaviour network it has constructed so far. The human teacher can then intervene in the trial to correct mistakes, effectively forcing the robot to alter its network. This observe-practise-correct learning methodology is the second major contribution of this work.

For Nicolescu and Matarič, the task is necessarily defined entirely through demonstrations, albeit generalised over a number of iterations. The task will eventually be learned correctly provided that the 'mean demonstration' covers task space sufficiently and correctly. Furthermore, goals are necessarily linked to behaviours; it is only possible to monitor and thus attempt to achieve a (sub-)goal if the agent already possesses some behaviour which results in that goal. The authors state of their system:

> "The spectrum of tasks learnable with our method includes all tasks achievable with the robot's basic set of behaviours. If the robot is shown actions for which it does not have any representation, it will not

> be able to observe or learn from those experiences ... we are not aiming at teaching a robot new behaviours, but at showing the robot how to use its existing capabilities to perform more complicated tasks." (Nicolescu and Matarić, 2007, p. 419)

In GTLF, the task metric could be similarly derived from a single source (if that is the only one available), but may also be partly defined by prior social knowledge or internal drives, for example. If other sources of task knowledge are available, then it may be possible for an agent using GTLF to assess the demonstrations themselves for accuracy. Goals in GTLF are perceptual states, which are *independent* of behaviours. This allows an agent to monitor goals before it has any idea about the behaviours necessary to achieve them. The association between behaviours and goals is made via the learning process as and when sufficient information has been gathered.

Ultimately, we believe there is very little to criticise when it comes to both the premise and results of the Nicolescu and Matarič system. The main question which GTLF raises is: why limit the system to co-ordinating behaviours? Elsewhere they document in detail that the system caters for both hierarchical and sequential control (Nicolescu and Matarić, 2002), something which GTLF also accounts for, although this has not yet been validated through experiment (see Section 3.5). Their behaviour networks can be nested within other behaviour networks allowing in principle the solution of arbitrarily complex tasks. Why not, then, start the hierarchy at 'ground level', and allow the *construction* of behaviours out of perceptual classes and action elements *as well* as the co-ordination of behaviours for the completion of complex tasks? The answer to this question is probably that both construction and co-ordination are quite hard enough problems in their own right (as both their results and ours indicate), without trying to build one on top of the product of the other (Bryson, 2001, ch. 11). However, in principle, GTLF does not have a cut-off point below which learning should not occur[5]. It would be interesting to test this principle in practise, and equally to see if there exists a level of description below which the Nicolescu and Matarić system fails to learn and / or control effectively. Also, using action-based rep-

[5]Of course, for any given implementation of GTLF there will be a minimum useful granularity of perception and action, but the point is that we impose no general bounds on this minimum.

resentations for the co-ordination of behaviours at higher levels of the GTLF[6] could be beneficial, while the construction of low-level behaviours may still be better suited to other representations (see Section 3.5.1).

Grollman and Jenkins

Another framework geared toward learning by demonstration is the Dogged Learning (DL) system of Grollman and Jenkins (2007). The novelty of their approach lies in the incorporation of *Mixed Initiative Control* (MIC) to provide the necessary task demonstrations. In most imitation learning systems, the demonstrator is another agent which the imitator either passively observes or actively follows / copies (Demiris and Hayes, 2002). Indeed, we have looked at one such system already. In DL's MIC policy, however, the demonstrator actually *assumes control* of the learning agent.

The DL algorithm itself is remarkably simple. Both the demonstrator and the imitator have access to the same sensory inputs. At each decision cycle, both agent policies are queried for an action to output along with a confidence value for its correctness. DL then uses the two confidence values to choose which of the proposed actions to execute, and then the imitator's policy is updated based on this input-output pair and the learning algorithm being used. In their experiments so far, Grollman and Jenkins have used a Sony Aibo as their learning platform, while the task demonstrations have been provided by hand-coded algorithms. The imitator's policy updates were calculated using Locally Weighted Projection Regression (LWRP) (Vijayakumar et al., 2005).

From a design principles perspective, DL is as similar a system to ours as we have found. Like GTLF, it exists as a platform-independent framework for testing and comparing different learning methods and agents. It is agnostic as to the nature of its inputs and outputs, which could range from raw sensor data / individual motor commands to high-level concepts / co-ordinated movements. However, as Grollman and Jenkins point out:

> "It is important that the perception and action concepts provided be sufficient for the desired task, as we cannot ask the robot to learn to do a task that it does not have the tools to perform."

[6] In our terminology, the construction of arbitration behaviours.

While we accept that sometimes an agent will be genuinely incapable of performing a task, the authors seem here to be implying that for DL, an inadequate characterisation of the task space could be the limiting factor. Like GTLF, the association between perception and action can be arbitrarily complex, dependent upon the learning algorithm (or behaviour representation for GTLF) used. However, GTLF has the added strength of allowing inter-episodic reconfiguration of inadequate representations (Section 3.4), and the hierarchical structuring of both perception space[7] (Section A.3.2) and the tasks themselves (Section 3.5.1).

Although both systems are agnostic to the choice of learning algorithm / behaviour representation, DL is designed for incremental real-time online learning and GTLF for the episodic learning system described above. There is no reason to suppose, however (and the authors themselves raise this point), that DL could not be adapted to run in a "quasi-batch" mode analogous to GTLF episodic learning. Conversely, as we have demonstrated, provided that the choice of learning algorithm is appropriate with respect to the resources available, GTLF can be made to run incrementally (see Section 3.4.1).

Although, in principle, DL is impressively platform-independent, we see MIC as the limiting factor with respect to the range of agents which could actually make use of the system, since both the imitator and demonstrator 'minds' must make use of the same body. The learners, then, must be able to periodically relinquish control of their body while remaining party to sensory input and motor output for training the DL algorithm. Demonstrator agents must either reside within the imitator's body, or be capable of teleoperating that body (see also Section 9.1.3). The former is limited to disembodied agents, such as the hand-coded programs used by Grollman and Jenkins. The latter could also be a program, or in principle any embodied agent with sufficient task knowledge and a control interface (i.e. humans). These restrictions have no effect on the scope of DL within the purposes for which it was designed, that is, as a tool to allow the teaching of robots by naïve users. However, GTLF, which is not restricted in these ways, can be used for this purpose, as well as potentially many others. For example, GTLF allows for MIC: since the imitator can observe the input-output sequence using it's *own* representations while the task is being demonstrated (be-

[7]Hierarchical perception, as described in Section A.3.2, could be seen as a kind of preprocessing of the input data. Therefore DL could, in principle, also take advantage of it.

cause its own body is being manipulated), this falls under imitation learning with a one-to-one correspondence map in place. Then, every cycle the demonstrator is in control corresponds to Learning (Stage 1), and every cycle the imitator is in control corresponds to Testing (Stage 3).

7.3.2 Virtual Task Learning

The use of what we have termed 'virtual agents' for research into task learning is not yet as widespread as the use of material robots (or virtual agents whose purpose is merely to simulate material robots) for this purpose. In turn, there exists a lack of examples of implemented general-purpose learning frameworks, like those described above, in virtual agents. Instead, we now look at two virtual task learning systems which have successfully combined multiple learning techniques for solving complex tasks in real-time; all key goals of GTLF.

Morales and Sammut

The first example we consider lies within the relatively specialised domain of flight simulation. The original work was carried out by Sammut et al. (1992), and was considered sufficiently significant to be republished in a relatively recent book on imitation (Dautenhahn and Nehaniv, 2002). The latest developments (Morales and Sammut, 2004), which we will discuss now, are of particular relevance to us. The approach can be summarised as follows:

1. Use *behavioural cloning* on data generated by humans completing flight tasks to acquire broad descriptions of the necessary behaviour.

2. Explore this narrowed policy space using reinforcement learning to find a (near-)optimal policy.

Behavioural cloning is the process of extracting control rules from recorded data in an attempt to replicate the policy of an expert (or experts). It therefore fits exactly within our broad description of imitation (Section 2.3.1).

Morales and Sammut used a high fidelity flight simulator which produced symbolic output relating to aircraft position, velocity, orientation, roll, pitch and yaw, as well as to objects 'visible' from the cockpit. The data were output as symbolic Prolog facts. Similarly, action commands could be sent to the aircraft's

ailerons, elevators, throttle, flaps and gear levers. The huge state-action space thus generated was discretised using a relational representation: *r-states* and *r-actions* (Morales, 2003). R-states are conjunctions of first-order predicates, derived from the flight simulator output, which partition the state space into classes. For example:

`distance_target(State, close)` $\wedge$ `orientation_target(State, left)`

is an r-state which would be satisfied by any `State` in which the target is close and to the left. R-actions are similarly defined as classes of action, e.g. `move_stick(y)` where `y` is a value along the y-axis of the control stick. Included in the r-action representation is an r-state which acts as the pre-condition for applicability of the action.

When an expert pilot carries out a flight task, the state is recorded along with each action command given. To construct a behaviour clone from this log, each state-action pair is processed in turn. If it is an instance of an existing r-action, then it is skipped; otherwise a new corresponding r-action is created. Effectively, this process eliminates all r-actions which were never instantiated during the demonstration. Since the observed flights are unlikely to cover all possible circumstances without discrepancy, an additional exploration phase was introduced prior to reinforcement learning. During this phase, the r-actions stored previously are used to try to navigate the aircraft into unseen and / or uncertain regions of the task space. The user is then queried for an action command, which is used to generate another r-action. An adaptation of Q-learning to r-space, rQ-learning (Morales, 2003), is then used with this restricted set of r-actions to converge upon an optimal control policy.

Although the task domain in which GTLF has been tested so far differs somewhat from the flight simulations of this project, our methodologies have much in common. Firstly, throughout the development of COIL and GTLF our imitation module has sought to extract a policy from a log of observed expert actions – behavioural cloning. The main difference here is the data source: ours was collected by a bot using its limited sensors situated within the environment, resulting in subjectivity in and partial observability of the state-action sequence, whereas the flight simulator logs this directly from the expert with no interference. The use of multiple complementary learning methods is another rare feature that we share,

and by collating our data we now have cross-domain results pertaining to the relationship between imitation and RL.

The flight simulator outputs symbolic knowledge as Prolog facts; Unreal Tournament sensors output symbolic knowledge as attribute-value pairs. As r-states are conjunctions of logical predicates, we defined perceptual classes as conjunctions of (disjunctions of) sensor conditions, but r-states form a partition of sensor space, whereas perceptual classes can nest and overlap. This gives us the option of using hierarchical perception (see Section A.3), where concurrently present perceptual classes compete over time for priority. For both systems it is assumed that an initial state space division is provided (e.g. user-defined prior knowledge), but through re-prioritisation and reconfiguration, GTLF can update this division if it proves unhelpful. In both systems 'actions' in fact represent classes of executable actions, which interestingly has led to exactly the same problems: how to choose a representative from the class to execute in practise, and how to stop 'jerky' behaviour resulting from a sequence of discrete movements (see Section 9.2.2). In other work from the same group, they have used regression models to produce smoother flights (Isaac and Sammut, 2003), and they cite this as future work for this project. Their results may be of direct application to GTLF.

Bielefeld and Dublin

As we explained at the start of Chapter 5, the development of COIL and ultimately GTLF has its origins in the particular problem we chose in order to investigate social learning; that is, enabling gamebots to imitate human behaviour. This section reports on the work of two groups who have focused on this very problem, and have recently begun to collaborate.

The first of these groups is based at Bielefeld, and their starting point was the problem of imitating the movement of human players. They chose Quake II (id Software, 1997) as their experimental platform, which is an FPS approximately contemporary to Unreal Tournament. The structure of the environment, range of actions available, nature of the tasks that can be set, and indeed most of the relevant features, are common to both games. One important difference is the ability of the Quake engine to record so-called demo files – a sequential record of the state of a given bot at each sensor cycle. Thousands of such files are available

for download on the internet, providing a wealth of human gameplay data for analysis. By applying data pertaining to absolute position and relative angle and distance to opponents to networks of MLPs, they were able to create reactive behaviours to control velocity and viewpoint. This allowed their imitator bots to traverse paths in a manner comparable to that of a human player (Bauckhage et al., 2003).

To make this motion more natural, their next step was to apply a Neural Gas clustering algorithm (Martinez et al., 1993) to the training data to form a topological map – that is, waypoints along these paths. They then used a combination of PCA and k-means clustering to find *action primitives* – co-ordinated sequences of movement – to allow the bots to move smoothly from one mode to the next (Thurau et al., 2004c). By applying differing potential fields across these waypoints, Thurau et al. (2004b) were able to motivate movement between them. The forces generated by the fields changed as the bots moved through a sequence of state classes, mostly associated with item pick-ups. In Quake, as in UT, pick-ups can be seen as goals, and so this behaviour appears to be goal-oriented or 'strategy-level' as opposed to purely reactive (Thurau et al., 2004a). They applied similar techniques to learn basic context-dependent weapon handling behaviour (Bauckhage and Thurau, 2004).

At this point the Dublin group began to publish their work in this area. Gorman and Humphrys (2005) took a step toward 'genuinely' goal-oriented behaviour by deriving a topology which necessarily places nodes at item pick-up points, using an adaption of Elkan's fast k-means algorithm (Elkan, 2003). By modifying the value iteration method of reinforcement learning, their bots were able to arbitrate between multiple weighted goals, as well as take into account goal (i.e. pick-up) transience. This work complemented that of the other group at this time – that is, an extension of the Bayesian imitation model of Rao et al. (2007) (Thurau et al., 2005) – and led to collaboration. The result: a hybrid system, with Dublin's goal arbitration mechanism integrated with the Bielefeld Bayesian motion model (Gorman et al., 2006a). The 'believability' of the bots using this system was subsequently assessed using a novel formal method (Gorman et al., 2006b).

Along another line of inquiry, the Bielefeld group have begun to look at the problem of the exponential increase in sample size necessary for training as the

input state space grows (Thurau and Bauckhage, 2005). They have so far tried (using 3D co-ordinate data as a test case) projection onto a lower-dimensional manifold through Locally Linear Embedding (Roweis and Saul, 2000), but their results were adversely affected by the sensitivity of embedding space to parameter selection. Dublin's other contributions include the QASE API (Gorman et al., 2007), a well-stocked toolkit to aid fast development of AI bots, and most recently moving onto combat behaviour, focusing on the issue of replicating human *inaccuracy* for believability (Gorman and Humphrys, 2007).

Overall, there are a few key differences which set this work apart from ours, and will hopefully bring new insights to both methodologies. As Gorman and Humphrys (2007) state, their work is motivated by 'human-likeness' and believability:

> "...this places us at odds with most of our counterparts in the field of robotics, who use imitation as a means of quickly attaining optimal performance; since we are primarily interested in producing realistically human behaviour, we often need to deliberately strive for the competent yet suboptimal."

For us, a task performance metric, as opposed to a believability metric, provides the measure of learning success, along with the more 'conventional' sub-goal of filtering out noise behaviour from demonstrations. Having said that, a believability metric could itself be seen as a task performance metric: one that specifies 'form' as well as 'function', and which we would therefore likely classify as an *action-level metric* (see Section 3.3.1). Since GTLF is more suited to learning at the program-level and above, perhaps the two approaches are in fact complementary, rather than being merely two different perspectives on the same problem.

At any rate, we believe the quest for believability has naturally led the authors to a 'controller's-eye-view' of imitation; in other words, their aim is to create a bot controller which mimics a human controller (we discuss this in the context of embodiment in Section 9.2). In contrast, we have taken an 'agent's-eye-view', in which a bot with local perception and action (and therefore analogous to any embodied agent) observes and imitates another bot. For example, the data used to train the Quake bots contained state which would not have been detected by other bots in the environment (i.e. health, armour, inventory, etc.), but is used

in the imitator's action selection policy. In our system, task error is primarily due to the partial observability of the demonstrator's decision space, as opposed to the deliberate replication of demonstrator error.

The use of x-y co-ordinates for both high-level goal navigation and low-level motion modelling renders all such learned behaviour map-specific. Also, randomly, dynamically or relatively positioned goals cannot be accounted for. By using a more egocentric data representation as we have done, it may be possible to use their techniques to learn map-independent control policies. On the other hand, navigation at the highest level is necessarily map-specific, so perhaps a dual approach would be most appropriate, with allocentric region / room goal representations and egocentric local control. In fact this may already be intended, as in their latest work on combat behaviour, Gorman and Humphrys (2007) transform the raw input data to approximate human perception prior to training.

Another distinction arising from our differing perspectives relates to data: volume, collection, representation and processing. As we have said, there are a lot of data available for Quake, and this is ideal for probabilistic learning algorithms which require significant training. However, in most imitation learning scenarios, even Unreal Tournament, accumulating that volume of data could range from laborious to impossible. This raises the question whether such techniques could practically be transferred to other domains. Also, most of the papers cited above refer to the importance of factors such as good parameter selection and correct sample size; one even to the extent that hyper-sensitivity led to inconclusive results (Thurau and Bauckhage, 2005). Although not necessarily as rigorous or well-founded, it should be beneficial for a system to allow for the kind of 'quick-and-dirty' one-shot learning that allows agents to learn a task from very few trials (e.g. Nicolescu and Matarić, 2007, above). As far as data representation is concerned, we believe that there is a useful middle ground between the 'classical AI' and 'pattern recognition' categories emphasised by the collective authors. In addition to our own, a number of the previously mentioned systems (e.g. Roy and Pentland, 2002; Morales and Sammut, 2004; Grollman and Jenkins, 2007), as well as cognitive architectures such as ACT-R (Anderson and Lebiere, 1998), combine statistical learning algorithms with sensory-motor 'symbols'. This potentially allows for both generality and cognitive reasoning beyond that which

is possible using sub-symbolic processes alone, without ever suffering from the symbol grounding problem of classical AI.

Differences aside, Bielefeld and Dublin are well on their way to achieving what we could not with COIL; that is, a fully integrated bot trained by imitation and capable of competing against humans. We believe this would represent a major milestone in imitation learning research.

Other Systems

The learning system of Le Hy et al. (2004) has elements in common with both ours, and that of Bielefeld and Dublin. Like us they used the GameBots API in Unreal Tournament, and were therefore subject to the same perception, action and data constraints. Their method is based on a Bayesian behaviour selection model. The bots start with a small set (six in their implementation) of hand-coded behaviours, such as *Attack*, *SearchHealth*, and *Flee*. In the model, the discrete random variable S_t represents the *behaviour state* of the bot at time t; that is, which of the base behaviours it is executing. They then specify seven other random variables corresponding to various relevant sensor states. To allow behaviour selection, conditional probability tables must be specified as follows:

- One for each sensor state conditional on the next behaviour state, $P(Sensor|S_{t+1})$, e.g. "what is the state of $Health$ likely to be, given that the next thing I'm going to do is $Flee$?"
- One for the next behaviour state conditional on the current behaviour state, $P(S_{t+1}|S)$ – "what should I do next, given that I'm currently $Flee$ing?"

The latter table is effectively a baseline state sequence model. The others, although somewhat counterintuitive due to the sensor state being conditional on the behaviour rather than the other way around, drastically reduces the number of input probabilities required. This technique of *inverse programming* is one of the major contributions of the paper. Instead of specifying these tables manually, they can be derived from human demonstration. Either the player simply directs, in real-time, which behaviour they want the bot to execute, or they play the game using a natural interface, and the behaviour state is inferred through recognition algorithms. Again, this system uses discrete behaviour symbols and sensor state

categorisations to allow probabilistic inference. It operates only at the 'strategy' / 'behaviour arbitration' level, with all lower-level behaviour assumed to be part of prior knowledge. Not only internal bot state (like the Bielefeld and Dublin system), but also in some cases the behaviour category choice is recorded directly from the demonstration, as opposed to via intra-environmental sensing. This places it at a similar distance with respect to our 'agent-centric imitation' ideal.

The argument for using FPS games as a research platform for advanced AI has perhaps been made most strongly by Laird and van Lent (2001). They also used Quake, with their bot AI being supplied by the SOAR cognitive architecture (Laird et al., 1987). SOAR has a native learning mechanism called *chunking*, which basically stores the solutions to previously solved problems, thereby eliminating the need for reasoning when a similar problem is encountered in the future. This kind of learning is weak inasmuch as it can neither produce novel behaviour nor organise existing behaviour – it simply reduces 'thinking' time. Chunking was disabled in the basic Quakebot, but was used to compile rules for predicting opponent actions by running internal simulations in later work (Laird, 2001a). More recently, they have added Reinforcement Learning capabilities to SOAR (Nason and Laird, 2005; Wang and Laird, 2007), which makes it yet another example of a system with symbolic reasoning coupled with statistical learning, although to date it has not yet been implemented in a Quakebot. A similar project called COGBOTS, this time using ACT-R (Anderson and Lebiere, 1998) in Unreal Tournament, seems not to have gotten past the conjecture stage (Lee and Gamard, 2003).

7.3.3 Summary

In this chapter, we have described in replicable detail an implementation of GTLF, and applied it to an investigation into multi-modal learning. Our results suggest that demonstrations which are good enough not to adversely affect an agent learning purely by observation, can nevertheless introduce enough doubt to those also using trial-and-error to cause unnecessary exploration and thus behavioural error. On the other hand, very bad demonstrations which leave parts of the task space unvisited can be compensated for by the exploration inherent

to trial-and-error.

We have also identified GTLF's nearest neighbours in the literature. The robot task learning system of Nicolescu and Matarić (2001, 2002, 2003, 2007) has proved very adept at learning novel high-level behaviour co-ordination, but may not have realised its full potential at the lower level of behaviour construction. Dogged Learning (Grollman and Jenkins, 2007) is a platform-independent framework like our own, but its reliance on Mixed Initiative Control somewhat limits its scope of application. The flight simulation work of Sammut et al. (1992); Morales (2003); Morales and Sammut (2004) is of great interest because they have encountered many of the same problems that we have, but in a very specialist domain inhabited by quite different agents. Finally, research by groups in Bielefeld and Dublin have created bots that successfully learn complex tasks in Unreal Tournament more successfully than we have been able to with COIL (Bauckhage et al., 2003; Thurau et al., 2004a,b,c, 2005; Bauckhage and Thurau, 2004; Gorman et al., 2006a,b; Gorman and Humphrys, 2005, 2007). However, we offer a different perspective on the problem: from inside the agent looking out, as opposed to from outside looking in.

In the final part of this dissertation, we continue to look outward into our research community, highlighting further the wide applicability of both GTLF and UT as a research platform.

Part III

Discussion

Chapter 8

Wider Applications

The experimental results presented in the previous chapter are primarily designed to convince the reader that GTLF is usable and useful. In this chapter we broaden our horizons, and ask two questions from different sides of the same applications coin. Firstly, what systems and methods exist that could contribute knowledge to and act as a foundation for GTLF? Secondly, how can GTLF be modified to contribute to research in the widest variety of learning domains?

8.1 Acquiring Prior Knowledge

Throughout our description of of GTLF theory (Chapters 2 and 3) and implementation (Chapter 7) we have been referring to prior knowledge. The term has encompassed all of the agent's innate abilities, as well as everything that it has learned up to the present time. GTLF makes use of a significant amount of such knowledge; the initial perceptual configuration and action repertoire for example. To allow us to focus on the task learning problem, we have so far assumed that all prior knowledge is provided by the agent's designers through hand-coding. In this section we hypothesise how some could be acquired using alternative means, and how these could be integrated with GTLF. Each reference to prior knowledge is examined in turn.

8.1.1 Initial Perceptual Configuration

This is the problem of defining an initial set of perceptual classes, and an accompanying function to map sensor space (determined by innate sensor configuration; see Section 3.1.1) onto this set. For the sake of this discussion, we assume that each sensor functions independently, and receives either discrete or real-valued data. One option is to use the perception channel system described in Sections 7.1.2 and A.3, assigning each sensor its own channel and each discrete input its own perceptual class. The remaining continuous sensor channels can then be subdivided arbitrarily, with each division mapping to a perceptual class. This is a valid initial configuration for GTLF, but may never converge to one that makes solving the task possible. That depends upon subsequent reconfiguration, should it occur (see Section 3.4).

If we suppose that the agent has had some opportunity to explore and interact with the world in which it lives, we can use that experience to automatically extract a better perceptual structure. There are many algorithms under the broad category of unsupervised clustering which attempt to detect statistical regularities in possibly high-dimensional data (Bishop, 2006). Pre-processing dimensionality reduction techniques such as Principal Components Analysis could also be used.

For examples in perception research, Gdalyahu et al. (2001) use a graph partitioning technique to cluster visual data, and Ajmera et al. (2004) use a Hidden Markov Model to cluster acoustic data. Each cluster could map to a perceptual class, and such an initial configuration should provide quicker convergence due to the extra information. Adding some degree of supervised learning could further reduce the amount of reconfiguration required during learning, but this increases the designer's knowledge input.

8.1.2 Initial Action Repertoire

This is the complementary problem of creating a set of executable action elements given a set of actuators. This in turn comprises two sub-problems: finding the relationship between commands sent and effects achieved, and co-ordinating different actuators to create useful elements.

The former is thought to be achieved by human infants through a process

known as 'body babbling' (Meltzoff and Moore, 1997). The infant sends exploratory motor commands to its actuators, and discovers the effect these commands have on its body configuration via perceptual feedback. The information and practise gained allow the infant to work toward forming goal-directed actions. An artificial analogue of this process could allow an agent to reduce the search space of possible action elements by eliminating those which have little or no (desirable) effect. How precisely this would be achieved in practise remains an open question.

The latter sub-problem has been of fundamental importance to roboticists trying to cope with the combinatorial explosion which occurs when creating controllers for robots with many degrees of freedom. Schaal et al. (2004) use both imitation and trial-and-error learning to find 'dynamic movement primitives' (DMPs), which are based on differential equations. Fod et al. (2002) apply PCA and clustering algorithms to human motion capture data (in a similar way as described above) to derive a set of 'perceptuo-motor primitives'.

8.1.3 Correspondence Library for Imitation Learning

Recall that in GTLF, the correspondence library is effectively a lookup table linking *egocentric* perceptual classes and actions to equivalent *allocentric* ones. For perceptual correspondence, this could be linking `to_my_right` to `to_expert_right`, and for action, `turn_right()` to `expert_turning_right` (see Section 3.1.3). Now we think briefly about how a library containing both perception and action correspondences might be constructed without resort to hand-coding.

Considering actions initially, what makes a good correspondence depends entirely on what level of imitation is required (i.e. action-level through effect-level; see Section 3.3.1), which in our case depends upon the task at hand. In other words, the task will determine the choice of error metric needed to measure degree of correspondence. We discuss work on generating such metrics automatically later in this section. For now, we assume that an appropriate metric has been decided upon, and that we are left with the problem of building a library. ALICE (Alissandrakis et al., 2002; Alissandrakis, 2003) and JABBERWOCKY (Alissandrakis et al., 2005), ALICE's successor specifically designed for imitating human motion, are capable of solving this problem. Like GTLF, ALICE is

a framework which allows different algorithms to be used to generate potential action correspondences. The system then checks to see if the generated action is a better correspondence (determined by the error metric) than the currently stored one, and updates the library if so. This is very similar to the testing stage of GTLF (see Section 3.3), but at a lower level.

At the effect-level end of the spectrum, action correspondence is more about satisfying equivalent goals, and therefore recognising goal-directed behaviour in others, than finding equivalent motor movements. Behaviour recognition has a significant literature of its own, so here we just highlight some applications that could potentially integrate with social learning in GTLF. No matter how the matching is carried out, 'recognition' implies that some set of model behaviours must already exist to match against. These could be user-defined or themselves learned from observation. Albrecht et al. (1998) have shown that a Dynamic Belief Network (DBN) can be used to predict a user's quest in a Multi-User Dungeon (MUD). Their approach is likely to be best suited to similarly discretely-defined domains. More applicable to UT is the work of Han and Veloso (2000), who represent dynamic real-time robot behaviour using Behaviour Hidden Markov Models (BHMMs). Probabilistic reasoning can then be used to find the most likely behaviour being executed given observations of a robot. Kaminka and Avrahami (2004) attack the same problem using a graphical representation of behaviour; more specifically, behaviour-based controllers. This allows them to use tagging and graph theory to generate behavioural hypotheses. Potentially the most challenging application of behaviour recognition is to 'natural' human activity. Again, it seems variations on the HMM are most popular (Clarkson and Pentland, 1999; Nguyen et al., 2003; Oliver et al., 2004), most likely because they can deal robustly with the uncertainty inherent when using noisy sensors (e.g. video and audio surveillance) and behaviour models (humans).

Perceptual correspondences can be formed through sharing perceptual contexts with another agent during completion of a task. This approach is used by Hayes and Demiris (1994), who experiment with an imitator robot following a teacher robot through a maze. The imitator's perceptual context at a given point where it must change direction corresponds to the teacher's perceptual context at that point. A similar method has been used to allow one robot to teach a lexicon to another (Billard, 2002). The teacher emits a different radio signal for

each perceptual class it occupies, and the following robot learns to associate the signals (symbols) with its own perceptual context at the time.

8.1.4 Lexicon for Instruction Learning

A lexicon for instruction learning serves much the same purpose as the correspondence library serves for imitation learning. This time the links would be between 'reference' perceptual classes (e.g. `heard_word_red`) and 'referent' perceptual classes (e.g. `see_colour_red`). Lexical learning is really just a specialised type of perceptual correspondence learning, so it is not surprising that one of the examples given above (Billard, 2002) relates to both applications. In that case, the references were (classes of) radio signals. Also, in Section 5.1 we examined in detail one of the best lexical learning systems to date: CELL (Roy and Pentland, 2002). In summary, using temporal and mutual information analysis, CELL finds 'central' examples of spoken words (the references) and visual stimuli (the referents). The perceptual classes are spheres in audio / visual feature space centred on these examples. The information gathered during interaction is also used to link word and stimulus classes to create a lexicon, which should be fully compatible with GTLF; our perception channel system (Sections 7.1.2 and A.3) is based on (and is in fact a generalisation of) that of CELL.

8.1.5 Reward Function for Trial-and-error Learning

Reward functions in trial-and-error learning determine the (real-valued) reward received by an agent in each possible state. This should not be confused with the *value function*, which gives the expected future rewards (*return*) for taking each possible action in each possible state, *given* the reward function (and a state transition probability function). Put simply, finding (or at least approximating) the value function is the aim of reinforcement learning (RL) (Sutton and Barto, 1998). Which RL algorithms are used will depend on the particular implementation of GTLF, but here we consider how the reward function itself, necessary for any RL algorithm, could be acquired.

The vast majority of RL applications make use of a human-designed evaluative feedback system, which gives rise to the *credit assignment problem*. One alternative is to use evolutionary methods, as demonstrated by Humphrys (1995,

1996). Here, Q-learning agents (Watkins and Dayan, 1992) with different reward functions encoded in their genomes compete for control of a robot via a process called *W-learning*. Groups of agents whose functions perform best with respect to some (hand-coded) fitness function are then selected for reproduction using a genetic algorithm (GA). A more recent example of related work is that of Damoulas et al. (2005a,b), where a similar process is described as 'evolving a sense of valency'. Clearly the extent to which these methods require repeated trial somewhat limits their practical applicability. Another alternative is to infer a reward function socially. For example, Atkeson et al. (1997) use human demonstrations of a pendulum swing to inform a function which rewards similar performance. Formally, the process of inferring an agent's reward function given its behaviour (and a model of the environment) is called *inverse reinforcement learning* (IRL) (Russell, 1998). Very recently, this has been extended to finding a probability distribution over the space of possible reward functions using the Bayesian framework (Ramachandran and Amir, 2007). The problem with these techniques is that they impose the same constraints as imitation learning: that the opportunity exists to observe an agent executing exemplary behaviour. In fact, IRL basically recasts the imitation learning problem in an RL framework.

8.1.6 Rules for Insight Learning

Any knowledge an agent has about the world and the way it operates (which could be helpful in task learning) should be represented by the rules in the insight learning module. Put another way, it is a 'catch-all' for types of learning or inference that don't fall into the other three categories.

We illustrate through an example from UT: suppose through reinforcement, instruction and / or imitation an agent learns that attacking enemies brings about a performance improvement in a given task. This does not, however, imply that the agent has any 'understanding' of the effect that its attack actions have on other agents – it cares only about improving performance according to the error metrics it is using. From the agent's point-of-view, there is no reason why attacking allies would not also be beneficial to task performance[1]. By the time the agent learns of its error, it could well be too late for a number of its

[1]Provided that it has not learned otherwise via the other three learning methods.

allies. If the agent has some insight into the situation, however, such as:

1. Attacking agents causes damage.

2. Causing damage to allies is bad.

Then fatal mistakes may be prevented. So how could such rules be acquired?

In a sense, finding rules is all that GTLF does. The task behaviours it generates can also be seen as hierarchies of *production rules*: 'IF <some perceptual state is occupied> THEN <initiate some course of action>'. So it may be that whichever learning algorithms are being used in GTLF could be adapted to learn 'insight' rules. For example, suppose that an MLP is being used to learn perception-action associations through imitation learning (Wood and Bryson, 2007a, and Section 6.2.1). A similar network could be used to associate current perception with past perception to produce basic cause-and-effect rules, for example. An obvious choice of algorithm for *rule induction* is the decision tree learner (C4.5 for example; Quinlan, 1992), since its inputs and outputs are generally already human-readable categories. However, *rule extraction* from neural networks using sub-symbolic data representation is also possible (Omlin and Giles, 1996), along with many other techniques (e.g. Cohen, 1995).

8.1.7 Error Metrics for Testing

Error metrics are used by agents to assess whether new behavioural tendencies learned in previous episodes have improved task performance. In Section 3.3.1 we adapted the concept of correspondence metrics for imitation, as defined by Nehaniv and Dautenhahn (2001), to create task performance metrics. Just as the choice of correspondence metric determines *what to imitate*, the choice of task metrics determine *what the task is*. In turn, deriving a correspondence metric automatically amounts to *learning what to imitate*, and deriving task metrics automatically amounts to *learning what the task is*. As was laid out in Section 2.1, a task is defined by a set of goals to be achieved. These goals could originate from three main sources:

1. **A passive observer** – that is, an intelligence external to the task environment. The agent designer is the most obvious example, who may in turn be representing society's 'corporate intelligence'. In this case, the agent would

be assessed according to pre-specified error metrics, defining the task according to some globally (i.e. socially) accepted standard. This is the case for all of our experiments.

2. **An expert presence** – an agent which possesses knowledge of a task and also inhabits the task environment. This knowledge could conform to social norms or be individual to the expert.

3. **Internal drives** – some agents may possess desires / drives / goals of their own, regardless of external input. In this case the agent can define its own success metrics, or maybe bias those that have been imposed upon it.

For Source 1, the learner has no way of inferring the metrics being used, since the observer is assumed to be outside of the agent's sphere of sensing. The only way for the learner to benefit in this case is if it is provided with the metrics in advance (i.e. as part of prior knowledge). We look instead at the more interesting cases in which it might be possible to infer error metrics. Where Source 2 is the best information source, task metrics would approach correspondence metrics. Put another way, if an expert demonstrator is the best source of information about a task, then performance in that task is best approximated by the quality of imitation achieved. *Learning what to imitate* and *learning what the task is* become the same problem. To this end, Calinon et al. (2007) use an improved version of a technique they developed previously (Billard et al., 2004) to determine the best metric (or *cost function*) for assessing imitative performance. Using a series of probabilistic processes (Gaussian Mixture Models [GMMs], Bayesian model selection, Hidden Markov Models and Gaussian Mixture Regression), a robot is able to extract high-level goals and thus reproduce two different task behaviours from observation of a human, even if the task environment changes. For inspiration from biology, Carpenter and Call (2007) give a good review of how animals and infants go about solving this problem.

In the absence of other sources, the agent must set itself a task, and this will be influenced by its internal drives and desires. A drive could be as simple as wishing to maximise some reward signal over a period of time. In this case, task metrics are like value functions (see above), where error is minimised for behaviour that is expected to elicit the highest returns. In principle, the complexity of an agent's drives, and the extent to which they could influence its choice of task metrics,

are entirely arbitrary and dependent upon the agent in question. It may also be that, rather than being the sole determining factor, drives merely weight already existing task metrics acquired from other sources.

Depending upon the experimental context, the inverse of the above problem may also need addressing. That is, how can an external observer assess an agent's task performance? If the observer provided the agent's task metrics in the first place (or has direct access to them), then there is a simple option – use the same metrics. If not, and this is always the case to some extent when studying animal behaviour, then the assessor has two options: either attempt to approximate the agent's task metrics via inference (the inverse of the above procedure), or assess according to some socially-defined norm. In both cases, but particularly the latter, it is possible that the agent and the assessor will use very different task metrics, and therefore make very different qualitative assessments of performance.

8.2 Into the Real World

So far GTLF has only been implemented in Unreal Tournament, where perception and action are noise-free. However, in Real World applications, this will not be the case, so it would be useful to see how GTLF could be improved to cope with noisy environments. Also, even in noise-free environments, partial observability may still introduce uncertainty into reasoning, particularly as perceptual classes become more complex.

8.2.1 Uncertain Perception and Action

Perceptual states in GTLF are like tiles covering sensor space. In probabilistic terms, given the sensor state, the probability of occupying a given perceptual state is either 0 or 1. That is not, of course, to say that perceptual class definitions are 'correct' in any absolute sense; boundaries can in fact be arbitrarily re-defined by the agent to maximise task performance (see Section 2.3.3). It just means that we have 100% confidence in our sensor readings — that they are entirely noiseless. If this were not the case, then it would make more sense to make use of a function which returns the probability of perceptual state occupancy given the sensor state. Put another way, rather than the perception system mapping

sensor states directly to perceptual states (see Section 3.1.1), it could map sensor states to probability distributions over perceptual states. These distributions would need to be determined either by known sensor noise models, or sampled from the environment.

The same reasoning applies to two other cases. Firstly, the case in which sensors are noiseless, but the sensor state does not contain all the information needed to fully determine the perceptual state: *partial observability*. In UT, an example of this would be the latent health of another bot, which can be estimated (i.e. assigned a probability distribution) based on past experience, but can never be read directly. Secondly, the case in which perceptual classes *themselves* have probabilistic as opposed to deterministic boundaries. For example, whether a light is on or off (i.e. the probability of occupying `light_on` or `light_off`) is near certain, but whether something is `blue` or `green` is a lot more subjective. Of course, there is also the possibility of noisy, partially-observable *and* subjectively defined task domains, common in the Real World, in which uncertainty from all sources would need to be consolidated.

If a given sensor state gives rise to a distribution over perceptual states, then this could be used as a basis for stochastic action selection. Assuming that skills remain as deterministic mappings between perceptual states and action elements, then the probability of occupying a given perceptual state could correspond to the probability of selecting the action element which is the image of that state under the skill function. Of course, even if sensor state deterministically maps to perceptual state (as in our experiments), action elements could still be chosen stochastically, using a softmax distribution for example (see Appendix A.2 for definition).

8.2.2 Summary

GTLF, like many complex learning systems, relies on a substantial amount of prior knowledge. However, there are a number of potential systems and techniques that could provide an alternative to laborious hand-coding. An initial perceptual configuration could be provided using statistical clustering techniques, (Bishop, 2006) as demonstrated for sound by Ajmera et al. (2004) and for vision by Gdalyahu et al. (2001). Fod et al. (2002) and Schaal et al. (2004) similarly

demonstrate methods for creating robot action primitives. Creating a correspondence library, particularly for agents with dissimilar embodiments, is the subject of work by Alissandrakis et al. (2002, 2005); Alissandrakis (2003), and the aforementioned CELL system (Roy, 1999; Roy and Pentland, 2002) implements lexical learning which could facilitate instruction. Humphrys (1995, 1996); Damoulas et al. (2005a,b) show that it is possible to evolve reward functions for Reinforcement Learning using a genetic algorithm. There are many rule induction methods (Cohen, 1995) which could seed the insight learning and perceptual reconfiguration modules, including the use of decision trees (Quinlan, 1992) and neural networks (Omlin and Giles, 1996). The technique used by Billard et al. (2004); Calinon et al. (2007) for learning what to imitate could potentially be adapted to create error metrics for learning the goals of a task. We also note that the Real World is subject to noise and uncertainty in a way that UT is not, and suggest that GTLF could be adapted for this by way of a probabilistic perception system and stochastic action selection.

Next, we turn to the more philosophical subjects of embodiment and correspondence in virtual domains.

Chapter 9

Wider Implications

In the previous chapter we looked outward, considering how our work might be integrated with other systems and implemented within other domains. Now we focus inward, aiming to promote our virtual agent research paradigm by giving our perspective on two topics discussed frequently in agent-based learning: *embodiment* and *correspondence*.

9.1 Intelligence, Embodiment and UT

Since the publication of the seminal Brooks (1991b,a) papers just over fifteen years ago, the idea that intelligence is fundamentally linked to embodiment has gained much support within the field of AI. Indeed, this principle is very much at the heart of the 'agent-based perspective' with which we align ourselves. It is beyond the scope of this thesis to give a systematic account of our opinions on the subject, taking into account the full philosophical and psychological perspectives. Instead we now attempt to tackle some important implications for us as AI researchers working in virtual domains.

It seems that some working in the field of Virtual Reality use the term 'embodiment' freely (Biocca, 1997). However, the emphasis of research such as this is more the realism and immersiveness of the environments — how much they give the user a sense of presence or 'situatedness'. Our perspective on Virtual Reality is slightly different, and looks to answer some of the following questions: is it possible to produce intelligent virtual agents? Can virtual agents be counted as embodied? Or is our research useful only insofar as it can be applied to material

domains?

9.1.1 Can one be embodied without having a body?

Although the term 'embodiment' has been used in many different contexts and research areas, to most it would imply materiality. Indeed, we have encountered those who vehemently insist that this is a requirement. This assumption has thankfully not gone unchallenged, however, and a number have attempted to abstract the concept of embodiment by analysing how it relates to agency and intelligence. We now trace one of these lines of argument through.

Etzioni

We initially look back to Etzioni (1993), who published a rebuttal to Brooks' papers two years later. Etzioni was working with *softbots*; software agents that 'inhabited' real-world software systems such as databases and the internet. In fact, current research on 'agents' often refers to this very type of agent. He claims that softbots offered 'easy embodiment', citing many of the same practical arguments that we do in defence of UT (see Section 5.2.2). His arguments are based on "software environments [which] are not idealisations of physical environments", in particular emphasising that such domains are neither designed nor controlled by researchers. He also states:

> "...in contrast to simulated physical worlds, software environments are readily available... and intrinsically interesting. Furthermore, software environments are *real*." (Etzioni's emphasis)

It is difficult to see where Etzioni would place Unreal Tournament in his thinking. It is not a specially designed research testbed; it is a 'readily available', commercially successful computer game designed to be challenging and entertaining (intrinsically interesting?). On the other hand, it is possible (just as it is in the Real World) to create highly constrained environments for experimentation. But is it *real*? He doesn't really define the term, nor does he attempt to pin down the notion of embodiment in any detail. For all these reasons, we need to look elsewhere for further support for virtual embodiment.

Kushmerick

A preliminary formal analysis of what it could mean for a software agent to be embodied came from Kushmerick (1997). When referring to agents, he also has in mind those which operate within real-world software environments. He is more impartial than Etzioni, however, who is openly defending his own agents from the implications of Brooks' claims. He introduces two important concepts for us: that of a *computational account* of the body, and that of *degree of embodiment*. For the former, he lists seven ways in which agents' bodies reduce the computational requirements of a task. We list two that he himself focuses on in his article:

1. Bodies exchange information with their environments across a high-bandwidth interface.

2. Bodies can off-load state into their environments.

Kushmerick criticises Etzioni's 'easy embodiment' claim, stating that (1) does not hold for his UNIX softbots. He argues that, since softbots cannot *broadcast* information (they send it [e.g. `rm`] point-to-point) or respond to *interrupts* (they must explicitly poll [e.g. `ls`] for sensory data), this epitomises *low*-bandwidth agent-environment interfaces. This judgement in turn leads him to concede that, because bandwidth is a continuum, so must embodiment be according to his framework. We discuss this further in a moment. For now, let us consider UT bots in light of Kushmerick's list.

Firstly, UT bots receive sensory data at a frequency of about 10Hz. This is a passive scan (i.e. it requires no polling by the bot) and contains data about 'continuously available' quantities, such as the bot's own health and ammunition status, and other visible agents and objects. Additionally, one-off events (such as nearby audible footsteps or taking damage) are received as bundles of extra 'interrupt' data. Bots can also initiate new actions at the same frequency, although some actions (such as jumping) have duration and cannot necessarily be interrupted. This gives bots a reaction time (depending on when in the sensor cycle an event occurs) of about 200ms, which is comparable to that of humans (again, depending on the task and many other factors). This makes sense, since UT bots are designed to behave in a 'human-like' way, as opponents to human players. The results of actions, such as a change in the bot's position or the

firing of a weapon, are 'broadcast' to nearby agents, and this happens automatically regardless of the agent's intent. Thus we can confidently say that according to Kushmerick's first criterion, UT bots are significantly more embodied than Etzioni's softbots.

Fulfilment of the second criterion is facilitated by the first, as the following example should clarify. The bot's ammunition, represented by a non-negative integer, is a variable which should greatly affect the bot's decision-making (e.g. fight or flight). As mentioned above, the bot receives its current ammunition status at each sensor cycle; the value is effectively 'held' externally by the environment (actually in the gamestate) and becomes known to the bot via its sensors. This is what Kushmerick means by 'off-loading state' into the environment. The alternative would require the bot to store an internal representation of its ammunition status, updating it as and when events occur which cause it to change. For ammunition, this may not be prohibitively costly, but the principle of trading off perception and representation applies to many aspects of UT. Bots' emphasis on perception, according to Kushmerick, implies a greater degree of embodiment.

Quick et al.

While Kushmerick's report focuses on exploiting the practical advantages conferred by embodiment within software domains, Quick et al. (1999) have a more bottom-up perspective (see also Dautenhahn et al., 2002). They propose the following minimal definition of embodiment:

> "A system S is embodied in an environment E if perturbatory channels exist between the two. That is, S is embodied in E if for every time t at which both S and E exist, some subset of E's possible states with respect to S have the capacity to perturb S's state, and some subset of S's possible states with respect to E have the capacity to perturb E's state."

It is minimal in the sense that it isolates embodiment, making no claims about its relation to intelligence or 'interesting' behaviour. By this definition, embodiment is independent of the nature of S and E, instead describing the relationship between them; a structural coupling of complex dynamical systems (Franklin,

1997). Since such a coupling has quantifiable properties, Quick et al. claim that this in turn makes degree of embodiment measurable. Although they suggest in passing that total complexity, as defined by Nehaniv and Rhodes (1997), could be a suitable metric, no actual quantitative analysis is given. Like Kushmerick, however, they do highlight two potential qualitative measures of embodiment:

1. *Perturbatory bandwidth* — 'total bandwidth of the perturbatory channels between system and environment, defined by the number and efficacy of the system's sensory and effector surfaces'.

2. *Structural variability* — 'structural complexity of system and environment, defined as the number of constituent components, and plasticity with regard to the configuration of those components in relation to one another'.

The former measure is closely related to Kushmerick's first criterion, which was discussed above. One further point: because UT is designed to be a game played from the bot's perspective, perturbation of the bot's state (S) with respect to its environment (E) 'matters' much more than the converse, and is therefore more likely to occur. For example, explosions which can 'kill' bots usually have no effect at all on the virtual masonry. As for the latter measure, Quick et al. basically add overall structural complexity to Kushmerick's second criterion, the ability to transfer complexity between system and environment (which they refer to as plasticity). In their papers they take an agent with very low variability and bandwidth, the E. coli bacterium, and 're-embody' it on the internet using words instead of chemicals to stimulate interaction (Quick et al., 1999). Following this principle, virtual UT bots, which ostensibly have substantially higher variability and bandwidth, can be seen as 'more embodied' than material bacteria. The problem with this assertion is that the state descriptions (i.e. components) of biological (and sometimes virtual) agents are very subjective and often determined by the agent's manifest behaviour. Maybe the behaviour is what matters, but making strong claims seems unwise when the subjectivity of the choice of metrics used for determining relative bandwidth and variability is also taken into account.

Riegler (2002) describes the above definition of embodiment as 'an important first step... [but] at the same time an insufficient characterisation'. He argues that since it applies to almost every conceivable system (even inanimate objects),

it is too general to be of practical use. He makes the stronger claim that 'a system is embodied if it has gained *experience* within the environment in which it has *developed*' (our emphasis). Although this leaves open the possibility of virtual embodiment, it is closed for any agent that has been designed rather than evolved: 'they are not embodied... but merely embedded in the dynamics of their environment'. For Riegler, cognition arises from embodiment due to evolution within an environment. Design replaces evolution and therefore makes embodied cognition impossible. It is unclear what his conclusions would be on agents which are partially designed and partially evolved (i.e. which have learned / developed / gained experience). Dautenhahn et al. (2002) respond with the underlying message that such criticisms are borne out of an attempt to make embodiment (and cognition, life, etc.) a binary property. Being 'too general to be of use' misses the point of the definition — not to classify but to measure degree of embodiment and allow for comparison between different system-environment pairs. Although they agree that cognition arises from embodiment, they see it as something which 'can be considered along a continuum of increasing degrees of embodiment'. In other words, cognition too is a matter of degree, and is strongly influenced by degree of embodiment. This raises another interesting question: how cognitive are UT bots?

Before we summarise the implications of these views for UT bots, it is worth noting that a broader overview of what embodiment can mean in different areas of research has been provided by Ziemke (2001, 2003).

9.1.2 Are UT bots embodied?

Throughout this discussion, it has been a continual temptation to be biased toward an affirmative answer to the above question[1], simply because UT bots *look* embodied. This is a clear credit to the game designers (Digital Extremes, 1999), as presumably this was exactly their intention. It is worth reminding ourselves, though, that UT is just a collection of bits and algorithms. Some of these elements represent bots (our system S), and some the environment (E). They are used by the graphics engine to create a 2D rendering containing human- and world-like images. In relation to this problem, Riegler (2002) makes a valid

[1]Quite apart from the benefit it would have on our research agenda.

point; that the 'designedness' of a system, environment, and relationship between the two can lead us to imbue meaning and generally over-anthropomorphise.

With this at the forefront of our thinking, can we still claim that UT bots are embodied? For every time t (bearing in mind that UT is a real-time game) a change of state in the environment E (i.e. an external event) can cause a change of state in the bot S. Similarly, a change in bot state (an action or internal event) can perturb the state of the environment. These mutual perturbations are facilitated by the algorithms associated with each set of variables. Also, all changes to a bot must come via the environment (or the bot's operator — see Section 9.1.3 below), whereas the environment can 'contain' many bots able to perturb different subsets of its state. We can thus view bots as being *embedded* in their environment (Dautenhahn et al., 2002) – the relationship is asymmetric, and the assignment of the labels S and E is not arbitrary. So, by Quick et al.'s minimal definition of embodiment, the easy answer to the question is 'to an extent'. However, a more appropriate question would be: to what extent are UT bots embodied?

Based on the properties of perturbatory bandwidth and structural variability, we are fairly safe in saying they are not as embodied as (most) humans. Bot sensors are not as numerous, sensitive or high-resolution as ours, and the human brain and nervous system is orders of magnitude more complex than the state space of a native UT bot, for example. We do, then, need to take careful account of how the lesser embodiment (and therefore lesser cognition?) of bots might affect our results before making any behavioural generalisations to highly embodied (and cognitive) agents such as humans. However, we suggest that UT bots are as embodied as many of the material robots used in social learning research to date. Many such robots operate in highly constrained environments and / or problems spaces to make learning possible (see Section 7.3.1 and Chapter 8 for examples). In practise, both types of agent can vary greatly in complexity, depending upon the resolution of internal representations used, for example. The point is that there is no reason in principle not to place social research carried out in UT on equal footing with that carried out using robots in the Real World. The two research domains theoretically provide the same opportunities and limitations for commenting on learning in general. They have the same issues of 'designedness', lesser embodiment and lesser cognition.

9.1.3 Embodiment and Teleoperation

So far we have implicitly considered UT bots which are operated by algorithms native to the UT engine — the intelligence is embedded in the environment with the agent[2]. However, the bots we have conducted experiments with have been operated remotely across a network, either by a human or a Java application. What effect does removing the intelligence from the environment have on embodiment? The philosophical nature of this question puts a complete treatment of it outside the scope of this thesis. Instead, we will informally discuss some possibilities assuming the more interesting case of human operation.

Firstly, the perturbation channels between body and environment are independent of the location of the mind, so removing the mind from the body still satisfies the minimal definition of embodiment. As far as degree of embodiment is concerned, the human visual and motor interfaces are designed to provide the same data and command options to a human that would be available to a UT-operated bot (see Section 9.2 below). Therefore, perturbatory bandwidth would change, because the sensory and effector surfaces would change, but maybe not drastically. However, by introducing the human brain into the control loop, structural variability could be seen to significantly increase, which would imply that human-operated bots are *more* embodied than native bots. This seems like a surprising result, but this notion of 'tele-embodiment', for some roboticists at least, is not an uncomfortable one (Paulos and Canny, 2001). Also, tele-embodiment is the only kind possible from the point-of-view of immersive Virtual Reality.

9.2 The *Real* Correspondence Problem

In Section 5.3.6, we mentioned the correspondence problem in relation to our COIL experiments, and saw the solution of mapping between two identical UT bots as a trivial one. But the section above suggests a deeper, nontrivial correspondence problem – that of mapping the perceptions and actions of one bot *operator* onto another. At this point we make an important distinction in terminology: the bot *operator* is the source of commands for the bot; the (possibly remote) intelligence or mind. The bot *controller* is a piece of software named

[2]What it would mean for the intelligence to be embedded 'within the agent' is unclear here.

as such within UT which provides an interface between the bot and its operator. The role of the controller is to translate whatever commands it receives and execute them (i.e. appropriately change the bot's state). Let us examine the scenario used in our COIL experiments in which a correspondence must be found between a bot operated remotely by an AI system and a bot operated remotely by a human.

9.2.1 Perceptual Correspondence

As far as perception is concerned, not much sensor data is made directly available to human players. Instead, the perceptual state required for decision-making must be extracted by interpreting a series of images on a screen. The images represent a projection of the 3D environment from the view point of the operated bot, as well as some on-screen data such as health and weapon status. Because the human brain can interpret these views similarly to views of the Real World, the perceptual classes used are also likely to be similar (e.g. `to_the_left` or `through_that_door`). The broad, stochastic nature of these classes could also be seen as resulting from the information loss between the game state variables and the brain (via the screen-eye interface).

An AI system, on the other hand, has direct access to the sensor data received by the operated bot, which arrives in bundles of attribute-value pairs. Such values could be of type real, integer, boolean or string symbols. In fact, the data perceivable via bot sensors is designed to match as closely as possible that perceivable via the screen, albeit represented very differently. For example, bot sensors only detect objects within their *view cone*, mimicking the restricted view provided on-screen. Also, bots cannot detect doors and walls (except through collision), but the human perception of navigating through discrete locations is approximated using a network of *way points*, which can be detected. For an AI system to learn socially from a human, it must be able to infer a *perceptual correspondence* that at least allows it to generate similar behaviour, regardless of how similar the underlying representations are (Bryson and Wood, 2005). GTLF is flexible in that it can use any combination of sensors and partitions to form perceptual classes. Good correspondences can in principle evolve via the representational reconfiguration process (see Section 3.4). For examples of possible correspondences, see

those we designed for our experiments (Sections 5.3.6 and 7.1.1).

9.2.2 Action Correspondence

The 'life-like' images UT displays give the human player a sense of presence in the environment, which results in 'life-like' perceptual classes. Action correspondences, however, are forced through a much more artificial process. The actions used by humans to operate bots are manipulations of the keyboard, mouse and / or joystick[3]. This input is then translated by the bot controller into executable actions.

An AI system can send commands to the bot controller using function calls. The two main differences in this method of interaction when compared to a human player are:

1. Function calls generally initiate actions which have duration, and therefore need a parameter which refers to a definite goal within the environment. In contrast, human players send continuous commands to the bot controller. The goals are managed moment-by-moment by the player.

2. As a result of this, there are 'short-cut' actions available via function call, which are not available to human players. Examples include `run_to(somewhere)` and `shoot_at(something)`. Of course, human players can replicate these actions, but they must do it using continuous commands.

In essence, the bulk of the action correspondence between human and AI results from both sets of action commands being forced through the bot controller. What remains is to infer a correspondence for the anomalies listed above. For AI attempting to learn socially from a human, this amounts to either forfeiting 'short-cut' actions in favour of more basic ones, or attempting to infer the goal of an action or action sequence. For simplicity, we chose the former for our experiments, but the latter could be made possible using action compilation rules (see Section 2.3.3) or machine learning techniques (Section 8.1.2). From a cognitive modelling point of view, there is some neurological justification of the capacity for recognising goal-directed action units (Rizzolatti et al., 2000; Hurley, 2005).

[3]If an immersive Virtual Reality interface was used, however, the action correspondences would become more 'life-like'.

9.2.3 Summary

The notion of embodiment for virtual agents is not an uncontroversial one. Brooks (1991b,a) seemed to rule it out in his seminal papers, but Etzioni (1993) disagreed, claiming that his *softbots* offered 'easy embodiment' since they resided in real world applications as opposed to simulated idealised worlds. Kushmerick (1997) gives a preliminary formal analysis of the concept, focusing on the practical benefits embodiment could have in software domains. This contrasts with the bottom-up perspective of Quick et al. (1999); Dautenhahn et al. (2002), who provide both a minimal definition of embodiment and add to it the notion of an embodiment continuum. By these definitions we conclude that UT bots are embodied to an extent at least as great as some of the material robots used in recent learning research, and should therefore be given an equal status as research tools. Also, as a consequence of so-called tele-embodiment, there exists a nontrivial correspondence problem when AI-controlled bots attempt to imitate humans in UT.

This concludes our account of the main contributions of this thesis. In the final part, we review these contributions and outline the directions we think this line of work would best take in the future.

Part IV

Conclusions and Appendices

Chapter 10

Conclusions

This dissertation contains descriptions of two large, complex learning systems, as well as numerous other algorithms, formalisms and experiments. In this final chapter we aim to tie all these threads together by summarising the entire dissertation and its contributions in a few pages. With these things at the forefront of the reader's mind, we then give some ideas for future research directions and draw the dissertation to a close.

10.1 Chapter Summary

Below is a brief description of the contents of each of the preceeding chapters for the purposes of both review and reference.

Chapter 1: Introduction

Intelligent agents need skills in order to handle the various tasks life in an unpredictable world throws at us. Not all skills can be gifted by evolution or design, but the remainder may be learned through practising tasks. We propose that this task learning process has elements common to all situated agents across many tasks. Identifying these elements and constructing a framework to house them is the primary purpose of this thesis. Such a framework should benefit engineers by giving them a structure within which to compare machine learning techniques, and should benefit scientists by giving them a starting point from which to create and compare biologically plausible models of task learning.

Chapter 2: An Agent-Independent Theory of Task Learning

We formulate an agent-independent description of task learning which rests on three fundamental concepts. *Perceptual classes* can represent any region of sensor space, from raw low-level data to complex high-level concepts. *Action elements* can represent any executable action, from low-level motor commands to high-level co-ordinated sequences of movement. *Skills* or *behaviours* map perceptual classes to action elements, and too can be implemented at any level. Armed with innate biases and past experiences to constrain and guide learning, agents can make use of a combination of *insight*, *trial-and-error*, *observation* and *instruction* to hone their skills, depending upon their capabilities and circumstances. For best long-term results, this is likely to occur over a number of *episodes*, with the learner aiming to gradually improve both task accuracy and efficiency.

Chapter 3: General Task Learning Framework

Based on the principles set out above, we can specify a General Task Learning Framework, which comprises four stages. In Stage 1, the agent explores the task environment according to the learning methods that are available, and that it wishes to use. Insight involves applying known rules and skill elements to the task; trial-and-error involves interacting with the task; observation involves watching an expert complete the task; and instruction involves attending to and interpreting a teacher. In Stage 2, knowledge gained during exploration is combined with old to create an updated skill and attention strategy. In Stage 3, the new skill is tested and feedback received from any agents monitoring the learner's performance. In Stage 4, the learner's perception space is reconfigured to enable improved accuracy in future episodes, and actions are compounded where possible to improve efficiency. The new skill is stored and the cycle iterates. Some tasks have inherent hierarchical or sequential structure which can be exploited to improve the efficiency of this process.

Chapter 4: GTLF as a Design Philosophy

By treating GTLF as a broad design philosophy as opposed to either an agent classification framework or baseline learning system, we find five considerations to put forward to the learning agent designer. Firstly, consider the whole problem,

as opposed to just the fragment of interest; define what is hard-coded so it is easier to define what needs to be learned. Secondly, consider the primitives being used; where they come from and how they might change. Thirdly, consider using different learning methods; recognise their relative strengths and weaknesses in different learning scenarios. Fourthly, consider your agent as one taken from a large design space; try and relate your agent to others. Fifthly, consider the bigger picture; although you might be interested in only a relatively constrained problem, try and hypothesise how the agent's operation could be generalised.

Chapter 5: COIL: Cross-channel Observation and Imitation Learning

GTLF and our interest in the characterisation of task learning grew out of a project which focused on imitation learning (Bryson and Wood, 2005). Specifically, our Cross-channel Observation and Imitation Learning (COIL) system (Wood and Bryson, 2007b) is a generalisation to imitation of Deb Roy's Cross-channel Early Lexical Learning system (Roy, 1999; Roy and Pentland, 2002). COIL channels and segments information relating to the perceptions and actions of an agent observed demonstrating a task. It then attempts to find good perception-action bindings by identifying co-occurrence, recurrence and high mutual information. The bindings are used as a specification for imitated behaviour which can then be executed by the learner. The system was implemented and tested in the virtual reality-style computer game *Unreal Tournament* (Digital Extremes, 1999). Agents (called bots) running COIL successfully learned two task behaviours from human demonstration.

Chapter 6: From COIL to GTLF

Despite COIL's initial success, we identified a number of potential problems looking to future development. Firstly, the representations designed for speech and vision inherited from CELL were not best suited for representing generic actions and percepts. Secondly, some of the algorithms inherited from COIL relied on assumptions which did not hold in the general case, and this adversely affected efficiency. Thirdly, COIL had no facility for representing hierarchical task or behaviour structure, which is likely to have rendered it unusable for complex tasks. We built an improved version of COIL which sought to address the first two

of these issues: simpler representations were used, and a generic MLP classifier network replaced some of COIL's native algorithms (Wood and Bryson, 2007a). Experimental results confirmed that the system had been improved, and we were also able to show that Bayesian techniques could be used to systematically introduce prior knowledge and inform attention. However, we still felt that working within the CELL framework was a limiting factor. We thus decided to take what we had learned in developing COIL and design a new framework which extended to other learning methods: GTLF.

Chapter 7: GTLF Implementation

As a proof-of-concept, we implement several GTLF modules and use the framework to carry out experiments investigating the relative merits of social (observation) and individual (trial-and-error) learning. To demonstrate compatibility with the Reinforcement Learning paradigm (Sutton and Barto, 1998), our trial-and-error learning module implements the Semi-Markov Average Reward Technique (SMART — Das et al., 1999). Our results suggest that demonstrations which are good enough not to adversely affect an agent learning purely by observation, can nevertheless introduce enough doubt to those also using trial-and-error to cause unnecessary exploration and thus behavioural error. On the other hand, very bad demonstrations which leave parts of the task space unvisited can be compensated for by the exploration inherent to trial-and-error. We also identify GTLF's nearest neighbours in the literature. The robot task learning system of Nicolescu and Matarić (2001, 2002, 2003, 2007) has proved very adept at learning novel high-level behaviour co-ordination, but may not have realised its full potential at the lower level of behaviour construction. Dogged Learning (Grollman and Jenkins, 2007) is a platform-independent framework like our own, but its reliance on Mixed Initiative Control somewhat limits its scope of application. The flight simulation work of Sammut et al. (1992); Morales (2003); Morales and Sammut (2004) is of great interest because they have encountered many of the same problems that we have, but in a very specialist domain inhabited by quite different agents. Finally, research by groups in Bielefeld and Dublin have created what we failed to with COIL; bots that successfully learn complex tasks in Unreal Tournament (Bauckhage et al., 2003; Thurau et al., 2004a,b,c, 2005; Bauckhage and Thurau, 2004; Gorman et al., 2006a,b; Gorman and Humphrys,

2005, 2007). However, we offer a different perspective on the problem: from inside the agent looking out, as opposed to from outside looking in.

Chapter 8: Wider Applications

GTLF, like many complex learning systems, relies on a substantial amount of prior knowledge. Other than laborious hand-coding, what systems and techniques exist which could supply this knowledge? An initial perceptual configuration could be provided using statistical clustering techniques, (Bishop, 2006) as demonstrated for sound by Ajmera et al. (2004) and for vision by Gdalyahu et al. (2001). Fod et al. (2002) and Schaal et al. (2004) similarly demonstrate methods for creating robot action primitives. Creating a correspondence library, particularly for agents with dissimilar embodiments, is the subject of work by Alissandrakis et al. (2002, 2005); Alissandrakis (2003), and the aforementioned CELL system (Roy, 1999; Roy and Pentland, 2002) implements lexical learning which could facilitate instruction. Damoulas et al. (2005a,b) show that it is possible to evolve reward functions for Reinforcement Learning using a genetic algorithm. There are many rule induction methods (Cohen, 1995) which could seed the insight learning and perceptual reconfiguration modules, including the use of decision trees (Quinlan, 1992) and neural networks (Omlin and Giles, 1996). The technique used by Billard et al. (2004); Calinon et al. (2007) for learning what to imitate could potentially be adapted to create error metrics for learning the goals of a task. Finally, we note that the Real World is subject to noise and uncertainty in a way that UT is not, and suggest that GTLF could be adapted for this by way of a probabilistic perception system and stochastic action selection.

Chapter 9: Wider Implications

The notion of embodiment for virtual agents is not an uncontroversial one. Brooks (1991b,a) seemed to rule it out in his seminal papers, but Etzioni (1993) was quick on the defensive, claiming that his *softbots* offered 'easy embodiment' since they resided in real world applications as opposed to simulated idealised worlds. Kushmerick (1997) gives a preliminary formal analysis of the concept, focusing on the practical benefits embodiment could have in software domains. This contrasts with the bottom-up perspective of Quick et al. (1999); Dautenhahn et al. (2002),

who provide both a minimal definition of embodiment and add to it the notion of an embodiment continuum. By these definitions we conclude that UT bots are embodied to an extent at least as great as some of the material robots used in recent learning research, and should therefore be given an equal status as research tools. Also, as a consequence of so-called tele-embodiment, there is a deeper correspondence problem than meets the eye when considering bots imitating humans in UT.

10.2 Review of Contributions

We now revisit and review the list of contributions made in Section 1.2.

The General Task Learning Framework

The General Task Learning Framework (GTLF) is a system for incorporating and investigating task learning in situated agents. It specifies a perceptual representation that can in principle interface with a huge variety of sensor configurations. It also specifies an action representation that is agnostic to the platform on which it is implemented. Skills are defined as executable maps from perception to action and can be applied at any level, or hierarchically at many levels, of task description. The framework itself is composed of many underspecified, independent modules which impose an order on information flow. The engineer need only implement those modules which are key to solving a problem, and can make use of the default algorithms provided both in this dissertation and in the Java implementation. The scientist can study the listed requirements of each module, together with the data flow, and use them as a basis for natural models of task learning. The combinatorial complexity of learning, and how it can be exchanged between different sub-systems, can also be studied. On paper, GTLF is suitable for implementing or modelling lifelong learning (Thrun and Mitchell, 1995) and autobiographical agents (Dautenhahn, 1998), since it allows for learning across multiple tasks and episodes, and for the gradual evolution of perception-action primitives. The validation of these more advanced properties through empirical testing provides an interesting path for future work.

The basic functioning of the framework *has* been validated in the virtual domain of Unreal Tournament, where it was used to investigate the effects of

combining different learning methods. It has been implemented in Java and is available, along with the code for our host agents, as a package of Java classes.

Cross-channel Observation and Imitation Learning

The Cross-channel Observation and Imitation Learning (COIL) system (Wood and Bryson, 2007b) is an adaptation of Deb Roy's Cross-Channel Early Lexical Learning system (Roy, 1999; Roy and Pentland, 2002) to learning by imitation. Like GTLF, it is platform-independent. COIL introduces the concept of using recurrence, co-occurrence and mutual information to form perception-action bindings which can compose novel behaviour. The extended version (Wood and Bryson, 2007a) demonstrates the novel use of *loss matrices* (Bishop, 1995, p.27) for rigorously encoding prior knowledge, and *automatic relevance determination* (Neal, 1996, p. 15) for selecting the most informative perceptual classes.

COIL has been validated through achieving successful imitation in UT. It has been implemented in Java and is available, along with its corresponding host agent code, as a package of Java classes.

Task Error Metrics

We have introduced a new formal methodology for measuring task error based on the approach of Nehaniv and Dautenhahn (1998, 2002) to measuring correspondence error during imitation. Our metrics use the task- and agent-independent perception and action representations specified in GTLF, and should thus be applicable in an equally broad variety of research contexts. *Action-level* metrics can be used to assess error in tasks which require precise, continuous movement in real-time. *Program-level* metrics can be used to assess error in tasks which must similarly be performed in a certain way, but at the level of function rather than form (Byrne and Russon, 1998). *Effect-level* metrics can be used to assess error in tasks which require only the satisfaction of (a sequence of) goals, and do not impose any constraints on method.

We demonstrate the utility of program-level metrics by using them to assess the performance of our learning agents.

JavaBots Extension

GameBots (Adobbati et al., 2001) is a module contained within Unreal Tournament which allows external programs to access the game environment and manipulate bots. JavaBots (Marshall, 2000) is a Java package that acts as a wrapper to GameBots, allowing for more rapid development of Java-based AI programs. We began developing extensions to this package while implementing a Java version of the BOD / POSH architecture (Bryson, 2001) for controlling UT bots (see Partington and Bryson, 2005; Brom et al., 2006, for related work). We have now added many classes and functions for various purposes, including parsing string input from sensors; systematically updating state variables with respect to the sensor cycle; regulating sensing and control; increasing bot spacial awareness; representing other bots, items and elements of the game map with Java-friendly objects; and (with Tristan Caulfield) executing automated trials. We have simply updated the JavaBots package, allowing the whole system to be ported to any version of Unreal Tournament which includes GameBots.

10.3 Limitations of GTLF

We have discussed many of the relative strengths and weaknesses of GTLF with respect to other systems in the literature review at the end of Part II. There are, however, two further issues which have surfaced a number of times during the development of GTLF which we now discuss.

Continuous Data

Perception in GTLF is based on a discrete unit: the perceptual class. In short, everything that gets as far as the learning core has to (in theory) have been filtered through the perception system and thus discretised. However there *are* times when perceiving a continuum of values is necessary, or at least seems natural. A reward signal received from the environment (or another agent) is a case-in-point.

The reason GTLF does not deal with continuous data is that its representations are tailored for skill-learning at the program-level and above. Continuous mapping at the action-level is simply beyond the scope of the system, although integrating GTLF with low-level learning algorithms is an interesting prospect.

Put another way, this means that fine-tuning skills and correcting behavioural errors is only possible down to the level of the finest possible (or sensible) categorisation. However, perhaps GTLF should be altered to allow continuous data through the perceptual system, as there are many potential uses for it other than in defining skills.

To this end, we hypothesise a new definition for the perceptual class. Instead of representing only a simple binary condition (i.e. whether some property is present or absent from the sensor state), a perceptual class could also encode some measurement retrieved from the sensor state (i.e. a real-valued function from sensor space to perception space). These values could represent strengths, rewards, intensities, distances, probabilities[1], etc., and could propagate throughout GTLF, potentially being used for more informed skill updates (especially during reinforcement learning), attention strategy updates, and in the reconfiguration of perception and action representations. The rate coding model (Rieke et al., 1999) provides an example of how such continuous information could be encoded neurally in biological (or bio-mimetic) agents.

Semantic Knowledge

The second issue is that of representing semantic knowledge. In short this is virtually impossible in GTLF; since everything is geared towards task learning, all of GTLF's data types have a distinctly procedural flavour.

For example, consider the fact: "soccer balls are round". It is perfectly possible for GTLF to contain a perceptual class for representing soccer balls, and even one for representing the notion of roundness. However, there is no natural way of connecting these two concepts using GTLF's native data types. Unfortunately, this kind of knowledge could come in very useful for task learning, particularly by insight. It could allow much better abstraction and transfer of knowledge between skills. This being the case, we would be wise to consider adding representations for semantic knowledge, as well as defining how this might interact with procedural knowledge, when designing future versions of GTLF.

[1] In fact, this could be a good way of representing uncertain perception (see Section 8.2).

10.4 Future Work

In this final section, we highlight what we believe to be the most promising avenues of research branching from this dissertation:

- Further implementation and validation of GTLF modules. As we have demonstrated, even a fairly rudimentary implementation can lead to interesting results. Specifically, we would like to see working modules for instruction, insight, and reconfiguration, as well as implemented examples of testing with our error metrics. We would also like to build a hierarchical task / skill, even if only initially for control and not learning, so that we can better understand the power of our representations.

- An implementation of GTLF for more complex skills. This is the other side of the coin: as well as seeking to validate more of the system, it would also be desirable to focus on its core elements and validate them more convincingly. The most obvious way to do this would be to use a minimal implementation of GTLF to learn a task significantly more complex than those described here, such as a full Unreal Tournament *Deathmatch*.

- Integration of GTLF with other agents and sub-systems. GTLF is not a complete agent architecture, and should not be expected to 'do everything'. It should, however, be integrable with other systems, some of which are conjectured in this dissertation. Fundamentally, we wish to validate our claim that GTLF can scaffold learning independently of the choice of host agent and task (provided learning is at the program-level or above).

- Further development of the task learning formalism. Just as Nehaniv and Dautenhahn's correspondence metrics have been the basis for a formal description of imitation learning, it may be that our task metrics could be a similar basis for task learning in general.

- Further investigation into the interplay between social and individual learning. It is still rare to find a system which incorporates more than one form of learning. This leaves much potential for new study, and our experiment has started just to scratch the surface of this potential.

- A programme of experiments investigating the effects and uses of loss matrices for encoding of prior action knowledge and Automatic Relevance Determination for selective attention when using MLP skill representations. We have so far only carried out initial experiments as part of the COIL extension.

- A comparative quantitative analysis of the degree of embodiment of UT bots. We would particularly aim to compare UT bot embodiment with that of state-of-the-art task learning robots. The complexity measure defined by Nehaniv and Rhodes (1997) could provide a suitable starting point.

10.4.1 Final Thoughts

We would like to thanks those who have taken the time to read this dissertation. We hope it will have inspired the reader to consider taking up or continuing the study of task learning. Any questions, offers of collaboration, or offers of funding would be gladly received.

Appendix A

Mathematical Notes

A.1 Notes on the Skill Function

In Section 2.1, we define a skill as a function:

$$s : \mathcal{P}(P) \rightarrow A$$

where P is the set of all perceptual classes, $\mathcal{P}(P)$ is the power set of P, and A is the set of all action primitives. Now, we do not expect skills to necessarily *explicitly* define mappings from every possible subset of P. Therefore, if we define $a_{null} \in A$, the *null action*, and $X \subset \mathcal{P}(P)$, the set of all subsets of P which remain unmapped by the skill, then we can let:

$$s(X) = a_{null}$$

and still have a legal function. In other words, all sets of perceptual classes not explicitly mapped to action space by a skill, are by default mapped to the null action. Also, the fact that a behaviour is *complete* (that is, defined on the whole of a given task space) does not necessarily mean $X = \emptyset$. For example, suppose that the perceptual class p_1 is ubiquitously present in a given task space, and suppose $s(p_1) = a_1$ for some action a_1. Then s defines a complete behaviour. But we could still have p_2, a perceptual class nested within p_1, for which $s(p_2) = a_{null}$, and therefore $p_2 \in X \neq \emptyset$.

A.2 MLP Details

In this section, we define some of the terms used in connection with the Multi-Layer Perceptron architecture described in Section 6.2.1.

1-of-c Encoding 1-of-c encoding is a way of representing purely categorical data. If there is a total of c categories, then category i is associated with a binary vector of length c in which only the ith bit is set. Example for four categories: $(1,0,0,0)$, $(0,1,0,0)$, $(0,0,1,0)$, $(0,0,0,1)$.

Cross-Entropy Error Function For a binary output vector $(y_1, y_2, ..., y_N)$, such as those used to represent action categories in our MLP implementation, the cross-entropy error function on network weights w is defined as:

$$E(w) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$$

where $(t_1, t_n, ..., t_N)$ is the binary vector representing the correct target category.

Softmax Activation Function Using a softmax activation function to the output layer of an MLP effectively forces the output values to sum to one; they can then be interpreted as posterior probabilities of category membership. The softmax activation for a given output node y_i is given by:

$$y_i = \frac{e^{a_i}}{\sum_{j=1}^{n} e^{a_j}}$$

Where a_i is the sum of the inputs to y_i from the previous (hidden) layer of nodes.

Scaled Conjugate Gradient Search Gradient descent can be used to update MLP network weights by seeking the minimum of an error function (in our case, the **cross-entropy error function** shown above). The optimal 'direction of descent' can be found by using *conjugate gradients*. *Scaled* conjugate gradients update the network weights iteratively after seeing every piece of training data and its corresponding error value, as opposed to

at the end of a batch of training data. The weight update is as follows:

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_n(w^{(\tau)})$$

where $w^{(\tau)}$ is the weight vector prior to update, $\eta > 0$ is the learning rate parameter, and $E_n(w^{(\tau)})$ is the error vector for training data point n.

A.3 Perception Channels in GTLF

Using the perceptual channel system described in Section 7.1.2, at most one perceptual class from each channel can apply at any given time. This allows us to decrease the average-case upper bound on the size of the perceptual state. With no structure, the upper bound is equal to $|P|$, the total number of discernible perceptual classes. With a perception channel system, this is reduced to n, the number of channels. In the worst case, every perceptual class is assigned to its own channel, and we have $n = |P|$, so we can say in general that $n \leq |P|$. How much less n is than $|P|$ depends on the task, the environment, the agent's sensors, and so on. For example, a task which requires monitoring a few different stimuli each with many different states will require few channels; a task which requires monitoring many stimuli each with few possible states will require many channels. This matches our intuitive notion of a perceptual information channel.

With n perception channels the perceptual state can be represented as an n-dimensional vector $(P_1, P_2, ..., P_n)$ with each element P_i representing the perceptual class contributed by channel i. If at a given instant no perceptual classes in channel i apply, then for completeness we say this channel occupies its *null class*, and represent this by $\emptyset_i$. If channel i contains m_i perceptual classes (including its null class), then there are a total of $m_1 m_2 ... m_n$ different perceptual states. This number grows polynomially with the average number of perceptual classes per channel, and exponentially with the number of channels.

As discussed in Section 3.2.1, a complete, tabular skill representation would require assigning an action element to every possible perceptual state: $m_1 m_2 ... m_n$ assignments in all. An alternative specification is to assign an action to every perceptual *class*, for a total of $m_1 + m_2 + ... + m_n$ assignments. This reduces the expressibility of behaviour, but scales far better than the specification above,

growing linearly with the number of classes and polynomially with the number of channels. However, there are still n perceptual classes which apply at any given time, and now one must be selected so that in turn a unique action can be selected for execution. This can be achieved through an attention strategy consisting of two elements: **perceptual prioritisation** and **perceptual hierarchy**.

A.3.1 Perceptual Prioritisation

For perceptual prioritisation, the simplest method is to order the channels: at a given instant, the channel with highest priority is inspected for its current perceptual class. If this channel occupies its null class, then the next in line in inspected[1] and so on. A disadvantage of this prioritisation is that it assumes all classes (except null) within a channel are of equal importance, which clearly may not be the case.

A more flexible method would involve ranking the perceptual classes individually. Only a partial order is required, since groups of classes from the same channel could have equal rank as they never compete. Storing priority information requires one piece of memory for each class, $m_1 + m_2 + ... + m_n$ in all, which has the same space complexity as action assignments for this method. Upon input of a perceptual state, the rank of each class is retrieved, compared and the highest selected. This takes n operations (one for each class), growing linearly with the number of channels. If either this search time or memory requirement turn out to be prohibitive, it is possible for an agent using this method to further select which channels to attend to.

For a tabular assignment, all channels *must* be attended to; otherwise the perceptual state is not well-defined and no action can be retrieved. Although it may in theory be possible for artificial agents to attend to an arbitrarily high number of inputs, it surely becomes impractical above a certain threshold. Certainly, even the most advanced biological agents have proven to have quite limited attention (see Section 2.3.2). Using perceptual prioritisation on the other hand, entire channels can be ignored by simply not examining the corresponding per-

[1]This assumes that a channel occupying its null class implies that 'nothing noteworthy' is occurring on that input. If this is not the case, then a new perceptual class should be formed to represent the noteworthy perception, which of course redefines the null class automatically. For completeness, some default action should be assigned to the null classes for execution when *all* of the channels have null input.

ceptual state element during decision-making. This could result in non-optimal behaviour, severely so if high priority classes are consistently ignored. To minimise this, a systematic way of governing attention is necessary.

A.3.2 Perceptual Hierarchy

One such system is to implement a perceptual hierarchy. As an agent develops a variety of skills for application in a variety of circumstances, the number of salient channels and perceptual classes could potentially increase without bound. However, each skill is likely to require only a few channels, leaving the rest inactive or redundant. Also, the granularity at which the environment needs to be monitored may change during a task, or if the agent is arbitrating between tasks. By nesting channels and perceptual classes, attention can be progressively focused on the most salient parts of the task domain. Using 'flat perception', every perceptual class points to an action to be executed when that class is 'in focus' (i.e. has the highest priority). In the hierarchical model, a class can point 'down the tree' to one of three things:

1. Another set of channels. These channels are independent of those above, and provide a new perceptual state vector. Typically they will relate to lower-level features of the parent class. The process of selecting a single perceptual class from this vector is identical to the parent process.

2. Another set of perceptual classes. This is effectively 'zooming in' on the parent class, and the new classes will be subdivisions of the parent. Since we are in the same channel, there will be no competing classes; one and only one of these new classes can apply, allowing the process to continue to the next level of detail.

3. An action. When a perceptual class which points to an action is selected, a leaf of the perceptual tree has been found and no further descent is required.

By this definition, 'flat perception' is simply a special case of hierarchical perception, in which all the top-level classes are leaves.

A.4 SMART and the *Somewhat* Semi-Markov Property

In the experiments described in Section 7.1.2 we use a Reinforcement Learning algorithm called *SMART* as the core of our trial-and-error learning module. However, we suggest that the task environment may not be fully Semi-Markov; here we explain the extent to which it does and does not fulfil this property.

Before we do this, however, we make the following observations. In some ways, it is not particularly important whether or not the task environment is Semi-Markov. The main point of the experiments in Chapter 7 is to demonstrate and validate the usefulness of the framework itself, and we never make formal claims about the results. Even if the task environment were completely non-Markov, the use of SMART could be seen as a poor choice of learning algorithm, but would not necessarily detract from the utility of the surrounding framework in any way. Secondly, the results show that SMART successfully learned the task asked of it to a greater extent. It is, of course, possible that had the task environment been fully Semi-Markov then SMART would have fully converged on the target behaviour within the allotted time.

Bear in mind that in order to fulfil the Semi-Markov property:

1. The state transition probabilities must depend only upon the action chosen in the current state, and not on any further state-action history (the Markov property).

2. The state transition *times* must similarly depend upon only the action chosen in the current state.

We now discuss these conditions with respect to the specific task learning problem from Section 7.1.2; the vial pick-up task. To assess the first condition, let us at first ignore times and focus simply on state transitions. The task learning problem in question *would be* Semi-Markov if the following statements held:

- When facing a vial
  - turning right will eventually mean there is a vial to the left.
  - turning left will eventually mean there is a vial to the right.

  - moving forward will eventually mean the vial is collected and disappears so that no vials are visible.*

- When a vial is to the left
  - turning left will eventually mean a vial is directly ahead.
  - turning right will eventually mean a vial is to the right with probability p, or no vials are visible with probability $1 - p$.**
  - moving forward will eventually mean no vials are visible.***
- When a vial is to the right, this is symmetric to the above case.
- When no vials are visible
  - turning right will eventually mean there is a vial to the right.
  - turning left will eventually mean there is a vial to the left.
  - moving forward will forever mean that no vials are visible.

However, the statements marked with stars do not hold:

* In fact, there is a non-zero probability that moving forward when facing a vial will take the agent back into a 'vial left' or 'vial right' state. This depends in some complex way on the process history; i.e. it is a non-Markov transition. A reasonably close Markov approximation would be to say that with some small but fixed probability p the agent will re-enter the 'vial left' or 'vial right' state, and with probability $1 - 2p$ the vial will remain ahead until it is collected.

** This is a more serious violation: in fact, whether or not a second vial is visible upon turning depends on the agent's position in the room, which in turn depends on the process history. From far back, two vials will be visible and the transition will be from 'vial left' straight to 'vial right'. Close up there will be a gap before a second vial is visible and the transition will be from 'vial left' to 'no vials visible'. The crude approximation given above is the best possible to a Markov transition.

*** This condition does not hold if the agent is between the wall and the vial, and facing in toward the other vials. In this case moving forward could cause a transition to either 'vial left' or 'vial right' depending on orientation. An agent would very rarely get into this situation; the best approximation to Markov would probably be just to ignore this case.

It is clear, then, that the task in question does not satisfy the first criterion of the Semi-Markov property. However, we have one more observation to make before commenting briefly on transition time. Situations ** and *** described above are entered less frequently the better the behaviour gets. Optimal behaviour will never enter these states. In other words, the closer the learner gets to convergence the 'more Markov' the learning becomes. The more it learns, the better it learns (situation * will always apply to a small extent).

If we were to accept that the state transitions alone could be described as *somewhat* Markov, then what about the transition times, to make the process *somewhat* Semi-Markov? As above, the *target behaviour* does have consistent time intervals given present and future state, so as learning converges the Semi-Markov property becomes gradually 'stronger'. In general, however, transition time depends on distance from the vials; further away means longer forward motion and shorter rotations; closer means shorter forward motion and longer rotation. In short, the second Semi-Markov condition is also violated. Since there is no formal definition of a *somewhat* Semi-Markov process, we leave the reader to decide how well this describes the task environment in question.

A.4.1 Darken-Chang-Moody Exploration

The Darken-Chang-Moody *search-then-converge* procedure (Darken et al., 1992) gradually decays the SMART learning and exploration rates to zero as the process continues. The nth update for both parameters (represented by Θ_n) is defined as:

$$\Theta_n = \frac{\Theta_0}{1 + \frac{n^2}{\Theta_\tau + n}}$$

where Θ_0 and Θ_τ are constants.

Appendix B

Glossary

Below are listed definitions of the key terms and abbreviations used throughout this dissertation, together with examples where appropriate. References to other glossary entries appear in bold.

A-Channel: [**Action Channel**] An A-channel in **COIL** receives data pertaining to a particular type of action being executed by an observed **expert**. If we suppose, for example, that an A-channel is created to monitor rotation in a given expert, then as the expert turns, the channel will change state.

A-Prototype: [**Action Prototype**] — see **A-Unit**

A-Space: [**Action Space**] In **COIL**, A-space is defined as the space of all possible A-**subevents** (i.e. action segment sequences).

A-Unit: [**Action Unit**] In **COIL**, an A-unit is an exemplary **A-subevent**, known as an **A-prototype**, plus a radius in **A-space** defining its boundary. Intuitively, an A-unit represents an exemplar of an action class, coupled with a defined scope for variation. For example, an A-unit representing a *jump* could have a jump of 1 metre at its 'centre' (i.e. as its A-prototype), and a radius (defined on jump length) of 0.5 metres. Anything falling within this radius falls inside the boundary of the A-unit and is matched as a *jump* (in this case perhaps leaving room for *hops* and *leaps* which fall outside).

Action Element: Just as **perceptual classes** can describe an **agent's** environment at many different levels of detail, so action elements can encode

an agent's interaction with its environment at many different levels. For example, action elements could represent low-level 'action primitives' (e.g. `straighten_arm()`, `bend_arm(90)`), compound actions (e.g. `serve(long)`, `smash()`), or even entire **skills** (e.g. `play_tennis_with(opponent)`, `play_badminton()`). We express the ability of agents to apply some action elements differently in different situations through the use of *action parameters*. Some elements may take no parameters (e.g. `walk()`), some may take numerical (e.g. `walk_forward(10)`), symbolic (e.g. `walk(backwards)`), or **deictic** (e.g. `walk_to(object)`). This action element terminology is derived from Bryson's definition of POSH reactive plans Bryson (2001).

Action-Level: We use the term 'action-level' to describe the lowest-level type of **task metric**; those which account for the exact movements executed during completion of a **task**. For example, if the task is to sign a sentence in sign language, then a metric which measures deviation in hand and digit position from the ideal position (that is, from the **target behaviour**) would be described as action-level. The term is borrowed from the imitation learning literature, where it is used to describes the precise reproduction of gestures, facial expressions, postures, orientations, and other fine-level movements.

Action Repertoire: An **agent's** action repertoire is defined as the set of all **action elements** it is (in principle) capable of executing. The potentially infinite variety of agent actions may be represented by relatively few action elements through the use of *action parameters* (see **Action Elements**).

Agent: Since the term 'agent' has come to mean a variety of things both in computer science circles and beyond, in this dissertation we make the following restrictions on the definition:

1. *Agents are situated in space and time.* We do not limit this to material or even Cartesian space, nor need time be continuous. Fundamentally, the agent and the environment it inhabits must be distinguishable.
2. *Agents perceive.* This includes both elements of the environment (exteroception) and its own state (proprioception).
3. *Agents act.* This includes both acting upon elements of the environment (actuation) and upon its own state (cognition). Actions take

time.

4. *Agents are resource-bounded.* Their perceptual resolution, memory and processing speed are all finite.

We tend to make reference to three main types of agent: *biological* (such as humans and animals), *robotic* (humanoid or otherwise), and *virtual* (such as computer game **bots**).

Allocentric: We describe **perceptual classes** as allocentric if they are described in terms of a reference frame external to the observer. This could be some fixed frame innate to the **environment** (such as compass directions) or centred around another **agent** (e.g. describing the percepts and actions of that agent). We have used the latter category of classes as part of the description of the imitation learning tasks listed in this dissertation.

Arbitration Behaviour: — see **Skill**

Attention Strategy: The role of an attention strategy is to select a subset of the full **perceptual state** available to the **agent** (i.e. *input selection*) to pass onto a **skill** (for *action selection*) or learning module. This effectively reduces the co-domain of the skill function being executed / learned which in turn decreases the complexity of action selection / learning. Of course, learning / applying an attention strategy will itself require some overhead, so ultimately there is a trade-off between the work done at this 'pre-processing' stage and the work done later.

BBAI: [**Behaviour-Based Artificial Intelligence**] BBAI was first developed by Rodney Brooks (1991b,a), and is characterised by modular intelligence. That is, intelligence decomposed into many, simpler, independent modules — **behaviours** — linked together by their inputs and outputs, but having no access to each other's internal state and no centralised arbitration. We borrow the idea of a behaviour or **skill** as a small unit of intelligence which can be arranged with others (possibly hierarchically) to complete complex **tasks**.

Behaviour: — see **Skill**

Bot: In **Unreal Tournament** (and other **First-Person Shooters**), the in-game **agents** controlled by humans and AI algorithms are known as bots. Bots are generally humanoid avatars whose movement is managed by a module in the game engine known as a *bot controller*. Any intelligence wishing to control bots must route control commands through this module, which effectively defines the bot's **action repertoire**.

C4.5: C4.5 is a decision tree learning algorithm designed by Ross Quinlan (1992), which uses normalised *information gain* (difference in *entropy*) to find optimal decision nodes amongst the attributes of the input data. The decision trees generated by C4.5 can subsequently be used to classify unseen data. The algorithm can handle both discrete and continuous data, missing attribute values, and can also perform automatic pruning of learned trees.

CELL: [**Cross-channel Early Lexical Learning**] CELL is a robotic learning system designed and created by Deb Roy at MIT, and described in his PhD dissertation (Roy, 1999; Roy and Pentland, 2002). In the associated experiments, a robot equipped with a camera and microphone is exposed to mother-child-like interactions with simple objects. From these interactions the robot learns a basic lexicon; that is, associations between visual features of objects (**S-Categories**) and spoken words describing those objects (**L-Units**).

Channel Set: In **CELL** and **COIL**, we define a channel set as the set of all channels of a given type, e.g. all **A-channels** or all **P-channels**.

COIL: [**Cross-channel Observation and Imitation Learning**] COIL is our adaptation of **CELL** to general imitation learning, implemented in **Unreal Tournament** (Wood and Bryson, 2007b). In our experiments, AI-controlled **bots** observe human-controlled bots carrying out simple tasks in real-time from within the **UT** environment. Using COIL, the observers acquire behaviours comparable (through the use of **task metrics**) to those of the so-called **expert** bots.

Complete Behaviour: We describe a **behaviour** as complete with respect to a **task** if it is everywhere-defined in the task space. In other words, for every

perceptual state that it is possible for the **agent** to enter, the behaviour specifies a way of choosing an action element to execute (i.e. could be stochastic).

Conspecific: Broadly speaking, a conspecific of a given **agent** is another agent of the same 'type'; exactly what this implies depends upon the context in which the term is used. In our experiments, we use it to refer to another **Unreal Tournament bot** of the same class; in biology it generally refers to two members of the same species; in other contexts it could refer to two completely identical agents, or merely two agents with a readily identifiable *perception / action correspondence* (e.g. two humanoids).

Deictic: A deictic variable is a variable which can refer to a different element of the environment or agent depending upon the situation; e.g. `nearest_opponent`, `damaged_servo`, `unexplored_room`. Some **action elements** may take deictic variables as parameters, which allows the actions to be applied in different situations; e.g. `dodge(nearest_opponent)`, `repair(damaged_servo)`, `explore(unexplored_room)`. In this way, a potential set of largely identical action elements (e.g. `repair_servo_1()`,..., `repair_servo_n()`) can be represented by a single 'multi-purpose' element.

Effect-Level: We use the term 'effect-level' to describe a type of **task metric** which takes into account only the **goals** (i.e. *effects*) of a **task**. The *process* by which these goals are achieved is disregarded. For example, if the task is to win a soccer match, then a metric which counts only the total number of goals scored could be described as effect-level. The term is borrowed from the imitation learning literature, where it describes the reproduction of goals by the imitator without (necessarily) the reproduction of either behavioural structure or precise motion.

Egocentric: We describe **perceptual classes** as egocentric if they are defined in terms of the perspective of the observing **agent**. For example, `nearest_object` and `turning_clockwise` are egocentric, since they use the observer as the reference point. Intuitively, egocentric classes should be useful when describing tasks which are centred around the learner or

participant, such as those included in the experimental work of this dissertation.

Event: In **CELL** and **COIL**, an event is associated with a **channel set** and has boundaries defined by some condition (or set of conditions) on *all* of the channels in that set. For example, the start of an event on a set of **A-channels** (that is, an *A-event*) could be triggered by the initiation of action from a state of inaction (in all channels); the end could be triggered by the cessation of that period of action.

Expectation Item: — see **M-E Item**

Expert: In our imitation learning experiments, an expert is an agent which provides a learning observer with a demonstration of the **skill** to be acquired / **task** to be learned. Despite the natural connotations of these terms, for us an expert need not necessarily act perfectly in accordance with a given task or skill, nor need its demonstration be overt or purposefully intended as such.

Environment: For our purposes, an **agent's** environment, which we also refer to ask the **task** environment, can perhaps best be described as the complement of the agent in the world. If the world which the agent inhabits can be seen as a vast array of arbitrarily complex state variables, then the agent will in some sense exercise control over or have 'ownership' of some subset of those variables. The environment is what's left over, which includes, amongst other things, other agents.

FPS: [**First-Person Shooter**] A First-Person Shooter is a genre of computer game in which the view displayed to the player on screen is designed to approximate the view of the avatar that the player is controlling. The games are played in real-time within often large and complex three-dimensional virtual reality-style arenas (or *maps*), and involve interacting with other game agents (or **bots**) and objects (or *pickups*). Simulated gun fights are commonly part of the gameplay, which is reflected in the genre's name. Examples include *Quake II* (id Software, 1997), **Unreal Tournament** (Digital Extremes, 1999), and *Half-Life II* (Valve, 2004).

Gamebots: — see **Unreal Tournament**

Goal: In GTLF, a goal is identified with a **perceptual state**; some state of the world, as perceived by the goal-setting agent, which it defines as desirable to occupy or pass through (for the purposes of completing a **task**, for example). To reiterate this point: we assume goals are always associated with agents and cannot exist independently of them.

Infimum: The infimum of a set X of real numbers, denoted $\inf\{X\}$, is defined as the *greatest lower bound* of that set. That is, it is equal to the greatest real number that is less than or equal to all the members of X. For example, $\inf\{1, 2, 3\} = 1$, $\inf\{x : x > 0\} = 0$, $\inf\{\mathbb{R}\} = -\infty$.

JavaBots: — see **Unreal Tournament**

L-Channel: [**Linguistic Channel**] An L-channel in **CELL** receives data pertaining to a particular stream of speech input. If we suppose, for example, that an L-channel is created to monitor a speech signal from a given agent received from a microphone, then as the agent speaks, the channel will change state.

L-Prototype: [**Linguistic Prototype**] — see **L-Unit**

L-Space: [**Linguistic Space**] In **CELL**, L-space is defined as the space of all possible L-**subevents** (i.e. phoneme sequences).

L-Unit: [**Linguistic Unit**] In **CELL**, an L-unit is an exemplary **L-subevent**, known as an **L-prototype**, plus a radius in **L-space** defining its boundary. Intuitively, an L-unit represents a spoken word, coupled with a defined tolerance for individual variation. For example, the word "ball" could be spoken using many different accents, pitches, volumes, etc.; an L-unit representing "ball" would account (to an extent) for these variations, but still exclude similar words (e.g. "bill").

Lexical Candidate: — see **Lexical Item**

Lexical Item: In **CELL**, a lexical item is defined as an **L-unit** paired with an **S-category**. Intuitively, this is a spoken word paired with some aspect of

an object that is described by that word; the word "round" and a round object such as a ball. The purpose of CELL is to generate lexical items automatically by observing infant-directed speech and object manipulation.

LTM: [**Long-Term Memory**] In **CELL** and **COIL**, LTM is the buffer which stores the final output of the system; **lexical items** for CELL, and **M-E items** for COIL. In other words, LTM is a dictionary, either of word-meaning pairs, or of perception-action pairs, respectively.

M-E Candidate: [**Motivation-Expectation Candidate**] — see **M-E Item**

M-E Item: [**Motivation-Expectation Item**] In **COIL**, an M-E item is defined as an **A-unit** paired with a **P-category**. Intuitively, this is an action paired with the perception that either caused or resulted from taking that action. In fact, we can differentiate M-E items on that basis: **Motivation Items** are those which represent perception that motivates a given action; **Expectation Items** are those which represent perception that is expected to follow a given action. Since Motivation Items can be interpreted as a map from perception to action, they can be used as a basis for building a simple reactive **behaviour**. This is the purpose of COIL; to automatically generate these items whilst observing another agent carrying out a task, so that the behaviour needed to complete that task can be reconstructed.

MDP: [**Markov Decision Process**] An MDP is a discrete-time stochastic control process which comprises the following elements:

- A set of discrete states.
- A set of discrete actions.
- A transition matrix which defines the probability of entering state s_2 in the next time step if action a is taken in s_1, for all possible actions and states.
- A reward function which defines the expected immediate reward of moving to state s_2 from s_1 having taken action a, for all possible states and actions.

The aim of the process is to maximise cumulative expected reward, and is related a *Markov chain* in that a **policy** learned in this way will take actions based only on the present state.

MLP: [**Multi-Layer Perceptron**] An MLP is a feedforward neural network model having at least one hidden layer of nodes (i.e. at least three layers in total). Nodes in an MLP have nonlinear activation functions and, through supervised learning, can be *trained* for use in pattern recognition and classification. There are many different training methods, activation functions, and error functions in common use; we describe those that we used in our experiments in more detail in Section A.2.

MTM: [**Mid-Term Memory**] In **CELL** and **COIL**, MTM is a buffer which stores pairs of subevents which recur within **STM**. At this stage, each subevent is a prototype for the channel set with which it is associated (**L-** and **S-prototypes** in CELL; **A-** and **P-prototypes** in COIL). Each pair, therefore, represents a candidate for the associative units being constructed (**Lexical candidates** in CELL; **M-E candidates** in COIL).

Motivation Item: — see **M-E Item**

Mutual Information: [**MI**] Mutual information is a measure of the reduction in uncertainty of one variable due to knowledge about a second variable. In COIL, MI is used to evaluate the degree of cross-channel structure captured by an **M-E Candidate** selected from **MTM** for a given configuration of radii. To calculate the mutual information for an M-E Candidate having **A-prototype** A_0 and **P-prototype** P_0, let R_a and R_p be random variables such that:

$$R_a = \begin{cases} 0 & \text{if } d_a(A_0, A) > r_a \\ 1 & \text{if } d_a(A_0, A) \leq r_a \end{cases} \tag{B.1}$$

$$R_p = \begin{cases} 0 & \text{if } d_p(P_0, P) > r_p \\ 1 & \text{if } d_p(P_0, P) \leq r_p \end{cases} \tag{B.2}$$

for each M-E Candidate (A-prototype A, P-prototype P) in MTM (excluding the selected one), given radii r_a and r_p. Then the mutual information

$I(R_a, R_p)$ for this configuration is:

$$I(R_a, R_p) = \sum_{i=0}^{1} \sum_{j=0}^{1} P(R_a = i, R_p = j) \log \left[\frac{P(R_a = i, R_p = j)}{P(R_a = i)P(R_p = j)} \right] \quad \text{(B.3)}$$

where the probabilities are estimated using frequency counts: the number of M-E Candidates in MTM that satisfy the criterion divided by the total number of M-E Candidates in MTM. For example, suppose that we have selected an M-E Candidate from an MTM containing 8, and have fixed radii r_a and r_p. Of the remaining 7 M-E Candidates, suppose that:

- 1 has both A- and P-prototypes which fall within r_a and r_p
- 3 have just their A-prototypes fall within r_a
- 1 has just its P-prototype fall within r_p
- 2 have neither A- nor P-prototypes which fall within r_a and r_p

The estimated probabilities, then, are as follows: $P(R_a = 0) = \frac{3}{7}$, $P(R_a = 1) = \frac{4}{7}$, $P(R_p = 0) = \frac{5}{7}$, $P(R_p = 1) = \frac{2}{7}$, $P(R_a = 0, R_p = 0) = \frac{2}{7}$, $P(R_a = 0, R_p = 1) = \frac{1}{7}$, $P(R_a = 1, R_p = 0) = \frac{3}{7}$, and $P(R_a = 1, R_p = 1) = \frac{1}{7}$. Substituting these into Equation B.3, we have $I(R_a, R_p) = 0.006$.

P-Category: [**Perception Category**] In **COIL**, a P-category is an exemplary **P-subevent**, known as a **P-prototype**, plus a radius in **P-space** defining its boundary. For example, a P-category representing *dark* could have the complete absence of any light at its 'centre' (i.e. as its P-prototype), with some predetermined small amount of light allowable under the same categorisation.

P-Channel: [**Perception Channel**] A P-channel in **COIL** receives data pertaining to a particular aspect of the perception of an observed **expert**, from the perspective of that expert. Suppose, for example, that a P-channel is created to monitor the position of other agents in the environment with respect to the expert. Then as the agents move around, or as the expert moves around and changes its perspective, then the state of this P-channel will change accordingly.

P-Prototype: [**Perception Prototype**]— see **P-Category**

P-Space: [**Perception Space**] In **COIL**, P-space is defined as the space of all possible P-**subevents** (i.e. perception segment sequences).

Perceptual Class: Formally, we define a perceptual class as some possibly disjoint (in fact, arbitrarily complex) subset of an **agent's sensor space**. We refer to the particular mappings an agent makes from sensor space onto perceptual space as the agent's **perceptual system**. Intuitively, a perceptual class could define anything from simple binary states (e.g. `switch_up`, `switch_down`), through complex percepts (e.g. `hear_footsteps`, `hungry`), cognitive entities (e.g. `opponent_is_aggressive`) and memories (e.g. `recently_saw_food`

Perceptual State: We define an **agent's** perceptual state as the set of all **perceptual classes** generated by the agent's **perceptual system** at a given time. In general, the perceptual state can contain overlapping classes (e.g. `red`, `car`, `red_car`) and nested classes (e.g. `object`, `vehicle`, `car`). In our GTLF experiments we demonstrate that by defining perceptual space in terms of n input **channels**, the perceptual state can be correspondingly defined as an n-dimensional vector (one for each channel).

Perceptual System: — see **Perceptual Class**

Policy: — see **Skill**

Program-Level: Program-level **task metrics** lie in between the **action-level** and the **effect-level** in terms of the kind of behavioural correspondences they account for. Some degree of behavioural structure (that is, the method or process by which the task is achieved) beyond merely the goals is captured, but the fine-level mechanics of movement is disregarded. In short, the focus is *function* as opposed to *form* or *effect*. For example, in a combat situation the goal may be to eliminate enemies, but the method used is also important. Eliminating innocents, needlessly wasting ammunition, and staying exposed to enemy attack should all be avoided, but the form of each individual action is unimportant. A metric which accounts for 'combat style' as opposed to just outcome could be described as program-level. Ultimately, any set of such program-level specifications could also be described

in terms of individual, competing effect-level metrics. The difference from the learner's perspective would be in attempting to replicate the behaviour as opposed to its effects. The former may be more efficient in a social learning situation where a behaviour model is present, but the latter may be the only option in the case of individual learning.

S-Category: [**Semantic Category**] In **CELL**, an S-category is an exemplary **S-subevent**, known as an **S-prototype**, plus a radius in **S-space** defining its boundary. Intuitively, an S-category represents some aspect of an object (i.e. colour, shape, etc.) with a defined scope for individual variation. For example, an S-category representing *round* might have a ball at its centre (i.e. as its S-prototype), but should also include a coin. An egg might be close to its S-space boundary, whereas a brick should fall outside.

S-Channel: [**Semantic Channel**] An S-channel in **CELL** receives data pertaining to some perceived semantic property of an object in the environment. Examples of visual semantic properties include different colours, shapes, textures, tones, etc. If we suppose, for example, that an S-channel is created to monitor the colour of a given object, then if the object changes colour, or if a new object of a different colour commands attention, then the channel will change state accordingly.

S-Prototype: [**Semantic Prototype**]— see **S-Category**

S-Space: [**Semantic Space**] In **CELL**, S-space is defined as the space of all possible S-**subevents** (i.e. object views).

Segment: In **CELL** and **COIL**, a segment is associated with an individual *channel* and has boundaries defined by some condition (or set of conditions) on that channel alone. The segment boundaries generated by a given channel are then extended across all channels within the same **channel set**; in other words, all channels within a set contribute segments to all other channels. For example, the start of a turn could trigger the start of a segment in an **A-channel** which monitors rotation, and this would therefore also start a new segment across all other defined A-channels.

Semi-Markov Decision Process: [**SMDP**] A Semi-Markov Decision Process is essentially a generalisation of the **MDP** to continuous time. Suppose we were to simply time-slice a continuous time process and treat it as an MDP. Then for any state-action combination with variable duration, the process becomes non-Markov; it depends *how long* an action has been executed for; it depends on state-action history before the present state. We can compensate for this by associating a state-time transition function with an MDP; this is an SMDP.

Sensor Space: — see **Sensor State**

Sensor State: An **agent's** sensor state is defined as the set of all readings from all functioning sensors at a given time; i.e. all of the available 'raw' sensor data. **Sensor space** is thus defined as the space of all possible sensor states. If we assume (as is possible with many agents) that the readings from a given sensor can be fully described in terms of a finite number of attribute-value pairs, then an agent's sensor state can be described as a finite-length vector with each index corresponding to a sensor attribute.

Skill: Formally, we define a skill as a function which maps sets of **perceptual classes** (or equivalently subsets of the **agent's perceptual state**) onto **action elements** (contained within the agent's **action repertoire**). Informally, a skill, combined with an **attention strategy**, defines a sequence of actions an agent should take in order to make progress in a **task** (a skill performs *action selection* after the *input selection* of an attention strategy). This definition of a skill is akin to the definition of a **behaviour** within the **Behaviour-Based Artificial Intelligence** community, and we tend to use the terms interchangeably throughout this dissertation (for those involved in Reinforcement Learning, a skill is analogous to a **policy**). Since skills are themselves a type of action element, skills can be organised into hierarchies. We refer to skills which map high-level perceptual classes onto other lower-level skills as **arbitration behaviours**.

STM: [**Short-Term Memory**] In **CELL** and **COIL**, STM is a buffer which stores pairs of subevents which have overlapped within a given time frame. In CELL, STM is implemented as a short queue (about 5 items) where the

pairs are L-subevents coupled with S-subevents (the basis of **lexical candidates**). In COIL, STM is a larger buffer containing pairs of A-subevents and P-subevents (the basis of **M-E candidates**).

Subevent: In **CELL** and **COIL**, a subevent is associated with a **channel set** and is defined as any consecutive sequence of **segments** across any subset of channels with that set.

Target Behaviour: We use the term 'target behaviour' to refer to a **behaviour** that an imitator would ideally learn to replicate by observing an **expert** carrying out (some variant of) that behaviour. We refrain from using the terms 'optimal' or 'correct', simply because a target behaviour could be entirely arbitrarily defined, having no necessary intrinsic reward, usefulness, or benefit to the **agent** (it could even cause harm).

Task: A task is some activity motivated by **goals** defined by some **agent** or group of agents. The nature of the goals determine how performance in that task is best assessed; that is, what type of **task metric** is most appropriate. Tasks tend to be ubiquitous for autonomous agents such as ourselves, and require the exercising of **skills** in order to be completed. This is the main motivating fact behind this dissertation.

Task Class: The **goals** of a task can be described at different levels of detail. If that level is fixed, then a broad range of individual tasks may be described by a single set of goals. For example, the sequence *get up*, *go to work*, *come home*, is likely to be satisfied every day by many different people doing many different jobs; i.e. carrying out many different individual tasks. The set of all tasks that is defined by a particular set of goals is an equivalence class which we call a task class.

Task Metric: A task metric defines a way of measuring how 'close' two **behaviours** are with respect to a given task. Where we have used task metrics in this dissertation, one of the behaviours has corresponded to a **target behaviour** for a given task, and the other has corresponded to an agent's learned behaviour for that task. In this case, the metric measures how well the agent has learned the task (up to that point). Task metrics

fall into three categories: **action-level**, which compares precise movement; **program-level**, which compares behavioural structure; and **effect-level**, which compares goal completion. Performance in a given task is likely to be more suitably assessed by one type of metric than another.

Tolerance Interval: A tolerance interval for a given measured quantity estimates the range of measurements that will with probability p contain a pre-specified proportion q of the population (the measurements are assumed to be normally distributed). This contrasts with a confidence interval which estimates the range in which *the measurement itself* falls. Suppose that we take a sample of size n of the measurement from the population. Then we can define a *two-tailed* tolerance interval as $\bar{x} \pm ks$, where $\bar{x}$ is the sample mean, s is the sample standard deviation, and k is the *tolerance factor*, calculated as follows:

$$k = \sqrt{\frac{(n-1)(1+\frac{1}{n})z^2_{\frac{1-q}{2}}}{\chi^2_{p,n-1}}}$$

where $z^2_{\frac{1-q}{2}}$ is the critical value of the normal distribution which is exceeded with probability $\frac{1-q}{2}$, and $\chi^2_{p,n-1}$ is the critical value of the chi-squared distribution having $n-1$ degrees of freedom which is exceeded with probability p. For more details, consult Croarkin and Tobias (2003).

UT: [**Unreal Tournament**] Unreal Tournament is a **First-Person Shooter** computer game (Digital Extremes, 1999). Its agents (or **bots**) can be controlled by humans, native AI algorithms, or external AI programs over a network via a UT plug-in module called **Gamebots** (Adobbati et al., 2001). **JavaBots**, which we extended, is a package of Java classes designed to interface with Gamebots allowing easier implementation of Java-driven bots.

Appendix C

Implementing JavaBots with GTLF in Unreal Tournament

We add this appendix so that exploring, using, and extending our code is as painless as possible for any that might wish to do so. We also hope that seeing how some of the pieces fit together in practise may aid in understanding the principles of the framework.

C.1 Requirements and Setting Up

The first step toward implementing an Unreal Tournament JavaBot is to make sure the necessary software is in place. We now briefly describe what is needed:

- *Install Unreal Tournament.* We used the *Game of the Year* (GOTY) version for Windows, which is available cheaply to buy online (any search on Amazon or eBay is likely to find several copies for sale). Like all commercial computer games, installation is simply a matter of inserting the CD and following on-screen instructions. There is also a Linux port of UT by Loki Software, Inc., but it is no longer maintained and apparently still requires the purchase of the Windows version (see `http://www.lokigames.com/products/ut/`).

- *Install GameBots.* Recall that Gamebots is an Unreal Tournament module that allows the control of bots by external programs over a network. The relevant files are available from the project sourceforge page: `http:`

`//gamebots.sourceforge.net/` — follow the *Downloads* link, and select the *UT-BotAPI* package. You then have a choice of how to install: either download the *.umod* file and double-click it — this will install the module automatically. Alternatively you can download the *.zip* archive and copy the *BotAPI.ini*, *BotAPI.int*, and *BotAPI.u* files into the *System* directory of your Unreal Tournament installation.

- *Install JavaBots, our extension classes and GTLF.* All of the Java code needed can be downloaded from `http://www.cs.bath.ac.uk/~cspmaw/disscode.zip` — simply unzip the archive into a location in your Java classpath. The packages of interest are:

  - *edu.isi.gamebots.client* — a slightly modified version of Andrew Marshall's JavaBot code.
  - *maw.gamebots* — including a number of utility classes, wrappers, example bots, etc. The most important class is *BasicBot.java* which extends Marshall's *Bot* class. Any bot designed for use in this setup should extend this class.
  - *maw.tls* — the GTLF classes as described below (Section C.3). We have also included all the test code in the *test* directory.

  The *BotRunnerApp* application can then be launched by typing "java BotRunnerApp" at the command line. To create a bot instance and connect it to your UT server:

  1. Start Unreal Tournament.
  2. Go to the *Game* menu and click *Start Practise Session.*
  3. Make sure you select one of the *Remote Bot* Game Types (i.e. one that uses the GameBots module), and select a map of your choice.
  4. Click *Start* — the game should start.
  5. Start the BotRunnerApp.
  6. Make sure the UT server's IP address is entered in the *Server* input box (you'll need to hard code this if you want to run the automated version — see below).

7. Go to the *Team* menu, select *Add Bot*, and then type in the name of the bot class in the *Class* input box (or select it from the drop-down menu if it is there) and click *Add Bot*.
8. Click *Connect All* — your bot should appear in the game.

So that we could run a series of automated trials, we have adapted BotRunnerApp: by typing "java BotRunnerApp -b<bot class name>" where <bot class name> is the name of the bot instance you want to create, the BotRunnerApp will open, create, and connect the bot automatically.

Having set up the platform, the next step is to create a test bot. We have included the code for such a bot below, along with explanatory comments.

C.2 Example: "Hello World!" GTLF JavaBot

As we mentioned above, the important class to override when designing new bots is the *BasicBot* class in the *maw.gamebots* package. The following code demonstrates both the methods that should be overridden to get a bot up and running, and also how to do a trivial set up of the GTLF. The latter requires creating a motor interface by overriding the *MotorInterface* class in the *maw.tls* package (see below).

C.2.1 HelloWorldBot

```
import maw.gamebots.*;
import maw.tls.*;
import java.util.*;

public class HelloWorldBot extends BasicBot{

    private TLS GTLF;

    // bot constructor
    public HelloWorldBot(){
        // call the superclass and set up the GTLF
        super("HelloWorldBot");
        setUpGTLF();
    }

    private void setUpGTLF(){
        // set up the root of the perceptual hierarchy
```

```
        ChannelSet perceptualSystem = new ChannelSet();

        // set up a channel which only has one perceptual class to
        // monitor self-existence
        Channel doIExist = new Channel("E");
        doIExist.add("y", "exist", "==", 1, "I exist!");
        perceptualSystem.add(doIExist);

        // create the GTLF, passing it the motor interface and
        // perceptual class definitions
        GTLF = new TLS(new HelloWorldBotMI(this), perceptualSystem);

        // create a (fixed) skill which will stipulate that the agent
        // should shout continuously
        Associations alwaysShout = new Associations(false);

        // define action elements
        maw.tls.Action shout = new maw.tls.Action("S");
        // and define the appropriate perception-action mapping
        alwaysShout.add(doIExist, "y", shout);
        // then add this to GTLF as the task behaviour
        GTLF.addTaskBehaviour(alwaysShout);
    }

    // this method is called at the end of every sensor cycle
    protected void onSyncEnd(double time){
        // set up a store for the sensor data
        TreeMap<String, String> sensorData = new TreeMap<String, String>();
        // if this is being called, then I must exist!
        sensorData.put("exist", "1");

        // then execute the appropriate action according to the
        // defined task behaviour
        GTLF.executeTaskAction(sensorData);
    }
}
```

C.2.2 HelloWorldBotMI

```
import maw.gamebots.*;
import maw.tls.*;

public class HelloWorldBotMI extends MotorInterface{

    // link to the bot's action functions
    private HelloWorldBot bot;

    public HelloWorldBotMI(HelloWorldBot bot){
        super();
        this.bot = bot;
    }
```

```
    public boolean execute(maw.tls.Action action){
        // the bot has been asked to execute a shout action
        if(action.getSymbol().equals("S")){
            bot.say("Hello World!", true);
            return true;
        }
        else{
            System.out.println("Action not recognised");
            return false;
        }
    }
}
```

C.3 Pseudocode

Below is some pseudocode which may help some gain a better understanding of GTLF, particularly any who would wish to implement it on a non-Java platform. We have listed only the main classes, and for each of those classes listed only the field names and most important methods. Basic 'housekeeping' methods such as constructors, mutators, and accessors are not described. Full java code is available for download at `http://www.cs.bath.ac.uk/~cspmaw/disscode.zip`.

C.3.1 TLS

This is the main class for the Task Learning System, which houses all of the subsystems; i.e. the perceptual system, the exploratory and learned behaviours, the learning modules, episodic memory, and the motor interface. The main methods are for executing exploratory or testing actions, and for passing the perceptual state onto the learning modules for processing.

```
class TLS {
    -- field list --
    ChannelSet perceptualSystem
    Associations taskBehaviour
    Associations exploratoryBehaviour
    Associations[] behaviourLibrary
    LearningModule[] learningModules
    EpisodicMemory EM
    MotorInterface motorInterface

    -- methods --
    get perceptual state given sensor state {
```

```
        use the perceptualSystem to look up the perceptual state for this sensor state
        and return it
    }

    execute exploratory action given sensor state {
        get perceptual state given sensor state
        return the exploratory action (defined in exploratoryBehaviour) for this perceptual state
        look up the action in the motorInterface and execute it
    }

    execute task action given sensor state {
        get perceptual state given sensor state
        return the task action (defined in taskBehaviour) for this perceptual state
        look up the action in the motorInterface and execute it
    }

    process incoming sensor state for learning {
        for each learning module defined in LearningModules:
            get perceptual state given sensor state
            pass the perceptual state on to the learning module for processing
            (this will update EM)
    }
}
```

C.3.2 ChannelSet

The perceptual system we implemented is the *perceptual hierarchy* model described in Section A.3.2. The root of the hierarchy is a set of perception channels, as described in this class definition. If a designer wishes to implement a different perceptual system, then this class can be overridden or replaced, as long as there exists a method for deriving the current perceptual state given the sensor state.

```
class ChannelSet {
    -- field list --
    Channel[] channels
    PerceptualClass[] order

    -- methods --
    get the highest priority perceptual class given the sensor state {
        for each perceptual class in salience order:
            if the perceptual class is present in the perceptual state:
                descend into it and further down the hierarchy
                return the highest priority leaf perceptual class
    }

    get the full perceptual state given the sensor state {
        for each channel in this channel set:
            get the perceptual classes which apply
```

```
        then return the set of all these classes
    }

    set the saliency of a perceptual class to a given value {
        look through the perceptual classes in this channel set
        set the saliency of the relevant perceptual class
        reorder the hierarchy at this level
    }
}
```

C.3.3 Channel

```
class Channel {
    -- field list --
    String symbol
    PerceptualClass[] perceptualClasses

    -- methods --
    get the part of the perceptual state defined in this channel (and below) {
        for each perceptual class in this channel:
            if this class is in the perceptual state,
                add it and descend deeper into the hierarchy
        then return the set of all these classes
    }
}
```

C.3.4 PerceptualClass

Note that the saliency, sub-features and sub-classes of a perceptual class are properties associated with the perceptual hierarchy model. These can be omitted or replaced if a different perceptual system is used.

```
class PerceptualClass {
    -- field list --
    String symbol
    SensorGroup[] sensorCriteria
    double saliency
    ChannelSet features
    PerceptualClass[] subClasses
    String description

    -- methods --
    is this perceptual class included in the perceptual state? {
        for each sensor group which defines this perceptual class:
            check that the conditions on that sensor group are satisifed
        if they all are, return true
        else, return false
    }
```

```
    return the highest priority perceptual class nested within this one {
        if this is a leaf class, return it
        otherwise, if this class has sub-features, look inside them
        otherwise, for each subclass of this class:
            look inside that subclass
        return the highest priority leaf class found
    }

    add this class and any subclasses to the perceptual state vector {
        if this is a leaf class, add it to the perceptual state vector being built
        and return the vector
        otherwise, if this class has sub-features, descend into them and add
        if appropriate
        otherwise, for each subclass of this class:
            if the subclass should be in the perceptual state,
                add it to the vector
        then return the vector
    }
}
```

C.3.5 SensorGroup

```
class SensorGroup {
    -- field list --
    SensorCondition[] sensorConditions;

    -- methods --
    is this sensor group satisfied given the current sensor state? {
        for each sensor condition which is part of this sensor group:
            check if the condition is satisfied
        if they all are, return true
        else, return false
    }
}
```

C.3.6 SensorCondition

In this implementation, a sensor condition defines the name of the sensor attribute for which the condition is specified, the type of comparison it should make (i.e. "<", ">", "=="), and the value with which the live input should be compared.

```
class SensorCondition {
    -- field list
    String sensorName
    String comparator
    String value
```

```
    -- methods --
    is this sensor condition satisfied given the current sensor state? {
        look up the value this sensor condition applies to in the sensor state
        use the appropriate comparator to check if the condition is satisfied
        if it is, return true, else return false
    }
}
```

C.3.7 Associations

The Associations class represents one of our implementations of a skill in GTLF. It consists of a map from perceptual classes to a list of actions ordered by associative strength. It also keeps track of the change in strength values for the previous learning cycle so convergence can be identified. It also contains a method for 'importing' a skill in utility matrix representation (see below).

```
class Associations {
    -- field list --
    Map<PerceptualClass --> ActionStrengthPair[]> associations
    Map<PerceptualClass --> double> changes
    Action defaultAction

    -- methods --
    increase the associated strength of a given action to a perceptual class {
        if the perceptual class already has associations:
            if the action is already associated to it:
                just adjust the strength
                otherwise, add an association
            normalise across all associations
            sort the associations by strength
            and record the changes made for convergence checking
        if the perceptual class has no associations
            first add the default action with maximum strength
            then call this method again with the same arguments
    }

    update this skill representation given a matrix of utilities {
        for each perceptual class in the utility matrix:
            if the mimumum utility value is negative for that class:
                shift all utilities for that class so that the least is zero
            for each action:
                add the utility value as an ActionStrengthPair to
                the associations list for this perceptual class
            normalise strengths for this perceptual class
            then sort by associative strength
        add this perceptual class and its new list to the associations map
    }
```

```
    check if learning has converged for this skill given an error threshold {
        if not all the necessary perceptual classes have associations
            then we haven't converged yet
        for each perceptual class in the associations map:
            if the action at the top of the list changed last time
                then we haven't converged yet
            if the last change was bigger than the error threshold
                then we haven't converged yet
        otherwise we have converged
    }
}
```

C.3.8 ActionStrengthPair

```
class ActionStrengthPair {
    -- field list --
    Action action
    double strength
}
```

C.3.9 Action

The Action class represents the GTLF action element.

```
class Action {
    -- field list --
    String symbol
    String[] parameters
}
```

C.3.10 LearningModule

LearningModule was an abstract class designed to be implemented by concrete learning modules. We ourselves implemented two: one for observation learning and one for SMART (trial-and-error) learning (see below).

```
abstract class LearningModule {
    -- methods --
    process incoming date from the perceptual system and update episodic memory { }
}
```

C.3.11 EpisodicMemory

Recall that in GTLF, episodic memory stores association data, perceptual selection data, and perceptual categorisation data. In our implementation, only the first two were catered for.

```
class EpisodicMemory {
    -- field list --
    Associations associations
    PerceptualClass[] perceptions
}
```

C.3.12 MotorInterface

Again, MotorInterface is an abstract class that needs to be implemented differently for every different type of agent. It provides the link between the abstract skill / action element representations and the concrete executable actions available for a given agent.

```
abstract class MotorInterface{
    -- methods --
    find out how to execute the given action element and execute it { }
}
```

C.3.13 Utilities

Utilities represents our other representation of a skill; as a matrix of perceptual class / action element utility values. It too contains a function for importing data from the Associations skill representation.

```
class Utilities{
    -- field list --
    PerceptualClass[] states
    Action[] actions
    double[][] utilityTable

    -- methods --
    get the (joint) highest utility action for this perceptual state {
        for each action value in the appropriate row of the matrix:
            if this is the (joint) highest value:
                add this action to the return set
        return the set of actions
    }

    update the utility matrix using values from an associations map {
        for each perceptual class in the associations map:
            for each action in descending order of strength:
                copy the strength value into the appropriate place in
                the utility table
    }
}
```

C.3.14 ObservationLM

These final two class definitions are included just for the sake of interest and completeness. They describe the two learning modules we have implemented in GTLF so far, starting with the observation learning module. This module contains a list of action channels (relating to the actions of the observed expert), a list of perception channels (relating to the perceptions of the expert), and two correspondence libraries; one linking expert actions to egocentric actions, and one linking expert perceptual classes to egocentric perceptual classes.

```
class ObservationLM extends LearningModule {
    -- field list --
    String[] actionChannels
    String[] perceptionChannels
    Map<String --> PerceptualClass> previousState
    Associations actionCorrespondences
    Map<PerceptualClass --> PerceptualClass> perceptionCorrespondences

    -- methods --
    process incoming data from the perceptual system and update episodic memory {
        check the action channels and update episodic memory given the perception channels
        save the current perceptual state
    }

    check the action channels and update episodic memory given the perception channels {
        for each action channel:
            if this action channel is represented in both the current
            and previous perceptual states:
                if the expert's action state has changed since last time:
                    update the association of this action state to the expert's
                    perception state
    }

    update the association of a given action state to the expert's perception state {
        look up the expert's action in the actionCorrespondence library to
        find the equivalent action for me
        for every perceptual class in the expert's perceptual state:
            look up the class in the perceptionCorrespondences library to find
            an equivalent perceptual class for me
            update episodic memory with a new association between the action and
            egocentric perceptual class
    }
}
```

C.3.15 SMART

SMART is the Semi-Markov Reinforcement Learning module we implemented for our experiments.

```
class SMART{
    -- field list --
    double[] parameters
    Utilities utilities
    PerceptualClass[] states
    Action[] actions

    -- methods --
    get the next action to execute given the current state {
        update the explore-exploit parameters
        select a random number between 0 and 1
        if the number is less than the explore threshold:
            execute a random exploratory action
        otherwise, execute the highest utility action for
        this perceptual state
    }

    update the rewards given the state, action and reward data {
        update the cumulative reward using the SMART equations
        update the average gain
        write the new reward to the appropriate place in the
        utilities matrix
    }
}
```

Bibliography

Adobbati, R., Marshall, A. N., Scholer, A., Tejada, S., Kaminka, G., Schaffer, S., and Sollitto, C. (2001). GameBots: A 3D Virtual World Test-Bed for Multi-Agent Research. In *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*.

Ajmera, J., Lathoud, G., and McCowan, I. (2004). Clustering and segmenting speakers and their locations in meetings. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'04)*, volume 1, pages 605–608.

Albrecht, D. W., Nicholson, A. E., and Zukerman, I. (1998). Knowledge acquisition for goal prediction in a multi-user adventure game. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 1–12.

Alissandrakis, A. (2003). *Imitation and Solving the Correspondence Problem for Dissimilar Embodiments – A Generic Framework*. PhD thesis, Department of Computer Science, Faculty of Engineering and Information Sciences, University of Hertfordshire.

Alissandrakis, A., Nehaniv, C. L., and Dautenhahn, K. (2002). Imitating with ALICE: Learning to imitate corresponding actions across dissimilar embodiments. *IEEE Transactions on Systems, Man, Cybernetics, Part A: Systems and Humans*, 32(4):482–496.

Alissandrakis, A., Nehaniv, C. L., and Dautenhahn, K. (2007). Correspondence Induced State and Action Metrics for Robotic Imitation. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(2):299–307.

Alissandrakis, A., Nehaniv, C. L., Dautenhahn, K., and Saunders, J. (2005). Achieving corresponding effects on multiple robotic platforms: Imitating in context using different effect metrics. In *Proceedings of the Third International Symposium on Imitation in Animals and Artifacts*, pages 10–19, University of Hertfordshire, Hatfield, UK. SSAISB.

Anderson, J. R. and Lebiere, C. (1998). *The Atomic Components of Thought*. Mahwah, NJ: Erlbaum.

Anderson, J. R. and Matessa, M. (1998). The rational analysis of categorization and the ACT-R architecture. In Oaksford, M. and Chater, N., editors, *Rational Models of Cognition*. Oxford University Press.

Arkin, R. (1998). *Behavior-based Robotics*. MIT Press.

Atkeson, C. G., Moore, A. W., and Schaal, S. (1997). Locally weighted learning for control. *Artificial Intelligence Review*, 11:75–113.

Baddeley, A. (2000). The episodic buffer: a new component of working memory. *Trends in Cognitive Sciences*, 4(11):417–423.

Baddeley, A. (2001). The concept of episodic memory. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 356(1413):1345–1350.

Barsalou, L. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences*, 22(04):577–660.

Barto, A., Sutton, R., and Anderson, C. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):834–846.

Bauckhage, C. and Thurau, C. (2004). Towards a Fair 'n Square Aimbot – Using Mixtures of Experts to Learn Context Aware Weapon Handling. In *Proceedings of GAME-ON 2004*, pages 20–24.

Bauckhage, C., Thurau, C., and Sagerer, G. (2003). Learning human-like opponent behavior for interactive computer games. *Pattern Recognition, Lecture Notes in Computer Science*, 2781:148–155.

Bekkering, H., WohlschlaÈger, A., and Gattis, M. (2000). Imitation of Gestures in Children is Goal-directed. *The Quarterly Journal of Experimental Psychology*, 53(1):153–164.

Billard, A. (2002). Imitation: A means to enhance learning of a synthetic protolanguage in autonomous robots. In Dautenhahn, K. and Nehaniv, C. L., editors, *Imitation in Animals and Artifacts*, Complex Adaptive Systems, chapter 11, pages 281–310. The MIT Press.

Billard, A., Epars, Y., Calinon, S., Schaal, S., and Cheng, G. (2004). Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2-3):69–77.

Biocca, F. (1997). The cyborg's dilemma: Embodiment in virtual environments. In Marsh, J. P., Nehaniv, C. L., and Gorayska, B., editors, *Proceedings of the Second International Conference on Cognitive Technology (CT'97), 'Humanizing the Information Age'*, pages 12–26, Aizu Wakamatsu City, Japan. IEEE Computer Society Press.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer Science+Business Media, LLC, first edition.

Bizzi, E., Giszter, S. F., Loeb, E., Mussa-Ivaldi, F. A., and Saltiel, P. (1995). Modular organization of motor behavior in the frog's spinal cord. *Trends in Neuroscience*, 18:442–446.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing: Algorithms, Architectures and Applications*, pages 227–236.

Brom, C., Gemrot, J., Bıda, M., Burkert, O., Partington, S., and Bryson, J. (2006). POSH Tools for Game Agent Development by Students and Non-Programmers. In *Proceedings of the Ninth International Computer Games Conference: AI, Mobile, Educational and Serious Games*, pages 126–133.

Brooks, R. A. (1991a). Intelligence without reason. In *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, pages 569–595, Sydney.

Brooks, R. A. (1991b). Intelligence without representation. *Artificial Intelligence*, 47:139–159.

Bryson, J. J. (2001). *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents*. PhD thesis, MIT, Department of EECS, Cambridge, MA. AI Technical Report 2001-003.

Bryson, J. J. (2003). The behavior-oriented design of modular agent intelligence. In Kowalszyk, R., Müller, J. P., Tianfield, H., and Unland, R., editors, *Agent Technologies, Infrastructures, Tools, and Applications for e-Services*, pages 61–76. Springer.

Bryson, J. J. (2005). Modular representations of cognitive phenomena in AI, psychology and neuroscience. In Davis, D. N., editor, *Visions of Mind: Architectures for Cognition and Affect*, pages 66–89. Idea Group.

Bryson, J. J., Lowe, W., and Stein, L. A. (2001). Hypothesis testing for complex agents. In Meystel, A. M. and Messina, E. R., editors, *NIST Workshop on Performance Metrics for Intelligent Systems*, pages 233–240, Washington, DC. NIST Special Publication 970.

Bryson, J. J. and McGonigle, B. (1998). Agent architecture as object oriented design. In Singh, M. P., Rao, A. S., and Wooldridge, M. J., editors, *The Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL97)*, pages 15–30, Providence, RI. Springer.

Bryson, J. J. and Wood, M. A. (2005). Learning discretely: Behaviour and organisation in social learning. In *Proceedings of the Third International Symposium on Imitation in Animals and Artifacts*, pages 30–37, University of Hertfordshire, Hatfield, UK. SSAISB, AISB.

Byrne, R. W. (2003). Imitation as behaviour parsing. *Philosophical Transactions of the Royal Society B*, 358:529–536.

Byrne, R. W. and Russon, A. E. (1998). Learning by imitation: A hierarchical approach. *Behavioral and Brain Sciences*, 21(5):667–684.

Calinon, S., Guenter, F., and Billard, A. (2007). On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(2):286–298.

Caro, T. M. and Hauser, M. D. (1992). Is there teaching in nonhuman animals. *The Quarterly Review of Biology*, 67(2):151–174.

Carpenter, M. and Call, J. (2007). The question of 'what to imitate': inferring goals and intentions from demonstrations. In Nehaniv, C. L. and Dautenhahn, K., editors, *Imitation and Social Learning in Robots, Humans and Animals*, chapter 7, pages 135–151. Cambridge University Press, first edition.

Chappell, J. and Kacelnik, A. (2002). Tool selectivity in a non-primate, the New Caledonian crow (Corvus moneduloides). *Animal Cognition*, 5(2):71–78.

Clarkson, B. and Pentland, A. (1999). Unsupervised clustering of ambulatory audio and video. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'99)*, volume 6, pages 3037–3040, Phoenix, Arizona, USA.

Cohen, W. (1995). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, Lake Tahoe, USA.

Croarkin, C. and Tobias, P., editors (2003). *NIST/SEMATECH e-Handbook of Statistical Methods*, chapter 7.2.6.3. NIST/SEMATECH. http://www.itl.nist.gov/div898/handbook/prc/section2/prc263.htm.

Custance, D. M., Whiten, A., and Fredman, T. (1999). Social learning of an artificial fruit task in capuchin monkeys (*cebus apella*). *Journal of Comparative Psychology*, 113(1):13–23.

Damoulas, T., Cos-Aguilera, I., and Hayes, G. (2005a). Valency as a mechanism for agent adaptation. In *Proceedings of Towards Autonomous Robotic Systems (TAROS 2005)*, London, UK.

Damoulas, T., Cos-Aguilera, I., Hayes, G., and Taylor, T. (2005b). Valency for adapative homeostatic agents: Relating evolution and learning. In *Proceedings of the Eighth European Conference on Artificial Life (ECAL 2005)*, Canterbury, UK. Springer-Verlag.

Darken, C. J., Chang, J., and Moody, J. (1992). Learning rate schedules for faster stochastic gradient search. In White, D. A. and Sofge, D. A., editors, *Proceedings of the 1992 IEEE Workshop on Neural Networks for Signal Processing 2*, pages 3–12, Piscataway, NJ. IEEE Press.

Das, T., Gosavi, A., Mahadevan, S., and Marchalleck, N. (1999). Solving Semi-Markov Decision Problems Using Average Reward Reinforcement Learning. *Management Science*, 45(4):560–574.

Dautenhahn, K. (1998). The art of designing socially intelligent agents: Science, fiction, and the human in the loop. *Applied Artificial Intelligence*, 12(7-8):573–617.

Dautenhahn, K. and Nehaniv, C. L., editors (2002). *Imitation in Animals and Artifacts*. Complex Adaptive Systems. The MIT Press.

Dautenhahn, K., Ogden, B., and Quick, T. (2002). From embodied to socially embedded agents – implications for interaction-aware robots. *Cognitive Systems Research*, 3(3):397–428.

Dayan, P., Kakade, S., and Montague, P. (2000). Learning and selective attention. *Nature Neuroscience*, 3:1218–1223.

Demiris, J. and Hayes, G. (1997). Do robots ape? In Dautenhahn, K., editor, *Socially Intelligent Agents: Papers from the 1997 AAAI Fall Symposium*, pages 28–30, MIT, Cambridge, Massachusetts. American Association for Artificial Intelligence Press, Menlo Park, California.

Demiris, J. and Hayes, G. (2002). Imitation as a dual-route process featuring predictive and learning components: A biologically plausible computational model. In Dautenhahn, K. and Nehaniv, C. L., editors, *Imitation in Animals and Artifacts*, Complex Adaptive Systems, chapter 13, pages 327–361. The MIT Press.

Digital Extremes (1999). *Unreal Tournament*. Epic Games, Inc.

Digital Extremes (2000). *UnrealEd 2.0*. Epic Games, Inc.

Drescher, G. L. (1991). *Made-up minds : a constructivist approach to artificial intelligence*. MIT Press, Cambridge, MA.

Elkan, C. (2003). Using the Triangle Inequality to Accelerate k-Means. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*.

Epstein, R., Kirshnit, C., Lanza, R., and Rubin, L. (1984). 'Insight' in the pigeon: antecedents and determinants of an intelligent performance. *Nature*, 308:61–62.

Etzioni, O. (1993). Intelligence without robots: a reply to Brooks. *AI Magazine*, 14(4):7–13.

Fod, A., Matarić, M. J., and Jenkins, O. C. (2002). Automated derivation of primitives for movement classification. *Autonomous Robots*, 12(1):39–54.

Franklin, S. (1997). Autonomous agents as embodied AI. *Cybernetics & Systems*, 28(6):499–520.

Gdalyahu, Y., Weinshall, D., and Werman, M. (2001). Self-organization in vision: Stochastic clustering for image segmentation, perceptual grouping, and image database organization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1053–1074.

Gorman, B., Fredriksson, M., and Humphrys, M. (2007). The QASE API - An Integrated Platform for AI Research and Education Through First-Person Computer Games. *International Journal of Intelligent Games and Simulations*, 4(2).

Gorman, B. and Humphrys, M. (2005). Towards Integrated Imitation of Strategic Planning and Motion Modelling in Interactive Computer Games. In *Proceedings of the Third ACM Annual International Conference in Computer Game Design and Technology (GDTW '05)*, pages 92–99, Liverpool.

Gorman, B. and Humphrys, M. (2007). Imitative Learning of Combat Behaviours in First-Person Computer Games. In *Proceedings of the Tenth International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games (CGAMES '07)*, Louisville, Kentucky, USA.

Gorman, B., Thurau, C., Bauckhage, C., and Humphrys, M. (2006a). Bayesian Imitation of Human Behavior in Interactive Computer Games. In *Proceedings of the International Conference on Pattern Recognition (ICPR '06)*, volume 1, pages 1244–1247. IEEE.

Gorman, B., Thurau, C., Bauckhage, C., and Humphrys, M. (2006b). Believability Testing and Bayesian Imitation in Interactive Computer Games. In *Proceedings of the Ninth International Conference on the Simulation of Adaptive Behavior (SAB '06)*. Springer.

Gould, J. (1975). Communication of distance information by honey bees. *Journal of Comparative Physiology A: Sensory, Neural, and Behavioral Physiology*, 104(2):161–173.

Graziano, M. S. A., Taylor, C. S. R., Moore, T., and Cooke, D. F. (2002). The cortical control of movement revisited. *Neuron*, 36:349–362.

Grollman, D. and Jenkins, O. (2007). Dogged Learning for Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2007)*, pages 2483–2488, Roma, Italy.

Hafner, V. V. (2005). Cognitive maps in rats and robots. *Journal of Adaptive Behavior, special issue on "Towards Artificial Rodents"*, 13(2):87–96.

Han, K. and Veloso, M. (2000). Automated robot behavior recognition. In Hollerbach, J. and Koditschek, D., editors, *Robotics Research: the Ninth International Symposium*, pages 199–204, London. Springer-Verlag.

Hayes, G. M. and Demiris, J. (1994). A robot controller using learning by imitation. In *Proceedings of the 2nd International Symposium on Intelligent Robotic Systems*, pages 198–204, Grenoble, France.

Herman, L. (2002). Vocal, social, and self-imitation by bottlenosed dolphins. In Dautenhahn, K. and Nehaniv, C. L., editors, *Imitation in Animals and Artifacts*, Complex Adaptive Systems, chapter 3, pages 63–108. The MIT Press.

Hershlag, N., Hurley, I., and Woodward, J. (1998). A Simple Method To Demonstrate the Enzymatic Production of Hydrogen from Sugar. *Journal of Chemical Education*, 75(10):1270.

Hillier, F. (2004). *Introduction to Operations Research*. McGraw-Hill Science/Engineering/Math.

Hinton, G., Dayan, P., Frey, B., and Neal, R. (1995). The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161.

Hölldobler, B. (1999). Multimodal signals in ant communication. *Journal of Comparative Physiology A: Sensory, Neural, and Behavioral Physiology*, 184(2):129–141.

Humphrys, M. (1995). W-learning: Competition among selfish Q-learners. Technical Report no. 362, University of Cambridge, Computer Science Laboratory.

Humphrys, M. (1996). Action selection methods using reinforcement learning. In Maes, P., editor, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB-96)*, pages 135–144, Massachusetts, USA.

Hurley, S. (2005). Active perception and perceiving action: The shared circuits model. In Gendler, T. and Hawthorne, J., editors, *Perceptual Experience*, chapter 6. Oxford University Press.

id Software (1997). *Quake II*. id Software.

Imanishi, K. (1957). Identification: A process of enculturation in the subhuman society ofMacaca fuscata. *Primates*, 1(1):1–29.

Isaac, A. and Sammut, C. (2003). Goal-directed learning to fly. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, pages 258–265. AAAI Press.

Kaminka, G. A. and Avrahami, D. (2004). Symbolic behavior-recognition. In *Proceedings of the AAMAS 2004 Workshop on Modeling Other agents from Observations (MOO 2004)*.

Kobayashi, T., Nishijo, H., Fukuda, M., Bures, J., and Ono, T. (1997). Task-dependent representations in rat hippocampal place neurons. *Journal of Neurophysiology*, 78(2):597–613.

Köhler, W. (1925). *The Mentality of Apes*. Routledge & Keegan Paul, London.

Kuris, A. and Norton, S. (1985). Evolutionary Importance of Overspecialization: Insect Parasitoids as an Example. *The American Naturalist*, 126(3):387–391.

Kushmerick, N. (1997). Software agents and their bodies. *Minds and Machines*, 7(2):227–247.

Laird, J., Newell, A., and Rosenbloom, P. (1987). SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64.

Laird, J. E. (2001a). It knows what you're going to do: adding anticipation to a Quakebot. In Müller, J. P., Andre, E., Sen, S., and Frasson, C., editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 385–392, Montreal, Canada. ACM Press.

Laird, J. E. (2001b). Using a computer game to develop advanced AI. *Computer*, 34(7):70–75.

Laird, J. E. and Rosenbloom, P. S. (1996). The evolution of the Soar cognitive architecture. In Steier, D. M. and Mitchell, T. M., editors, *Mind Matters*. Erlbaum.

Laird, J. E. and van Lent, M. (2001). Human-level AI's killer application: Interactive computer games. *AI Magazine*, 22(2):15–25.

Lawrence, S. and Giles, C. (2000). Overfitting and neural networks: Conjugate gradient and backpropagation. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 114–119.

Le Hy, R., Arrigoni, A., Bessière, P., and Lebeltel, O. (2004). Teaching bayesian behaviours to video game characters. *Robotics and Autonomous Systems*, 47:177–185.

Lee, F. and Gamard, S. (2003). Hide and Seek: Using Computational Cognitive Models to Develop and Test Autonomous Cognitive Agents for Complex Dynamic Tasks. In *Proceedings of the 25th Annual Conference of the Cognitive Science Society*.

Littman, M., Dean, T., and Kaelbling, L. (1995). On the Complexity of Solving Markov Decision Problems. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 394–402.

Louie, K. and Wilson, M. A. (2001). Temporally structured replay of awake hippocampal ensemble activity during rapid eye movement sleep. *Neuron*, 29(1):145–156.

MacKay, D. J. C. (1992a). The evidence framework applied to classification networks. *Neural Computation*, 4(5):720–736.

MacKay, D. J. C. (1992b). A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472.

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Marshall, A. N. (2000). *JavaBots*. University of Southern California, Information Science Institute.

Martinez, T. M., Berkovich, S. G., and Schulten, K. J. (1993). Neural gas network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569.

Matarić, M. J. (1997). Behaviour-based control: examples from navigation, learning, and group behaviour. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2):323–336.

McClelland, J. L., McNaughton, B. L., and O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights

from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457.

Meltzoff, A. and Moore, M. (1977). Imitation of facial and manual gestures by human neonates. *Science*, 198(4312):75–78.

Meltzoff, A. N. and Moore, M. K. (1983). Newborn infants imitate adult facial gestures. *Child Development*, 54:702–709.

Meltzoff, A. N. and Moore, M. K. (1997). Explaining facial imitation: A theoretical model. *Early Development and Parenting*, 6:179–192.

Mennella, J. A., Jagnow, C. P., and Beauchamp, G. K. (2001). Prenatal and postnatal flavor learning by human infants. *Pediatrics*, 107(6):e88.

Minsky, M. (1986). *The Society of Mind*. Simon & Schuster, Inc. New York, NY, USA.

Morales, E. (2003). Scaling up reinforcement learning with a relational representation. In *Proceedings of the Workshop on Adaptibility in Multi-agent Systems*, pages 15–26.

Morales, E. and Sammut, C. (2004). Learning to Fly by Combining Reinforcement Learning with Behavioural Cloning. In Greiner, R. and Schuurmans, D., editors, *Proceedings of the Twenty-first International Conference on Machine Learning (ICML 2004)*, pages 598–605, Banff, Alberta, Canada. ACM Press.

Nason, S. and Laird, J. E. (2005). Soar-RL, Integrating Reinforcement Learning with Soar. *Cognitive Systems Research*, 6(1):51–59.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer.

Nehaniv, C. L. and Dautenhahn, K. (1998). Mapping between dissimilar bodies: Affordances and the algebraic foundations of imitation. In Demiris, J. and Birk, A., editors, *Proceedings of the European Workshop on Learning Robots*.

Nehaniv, C. L. and Dautenhahn, K. (2001). Like me? – measures of correspondence and imitation. *Cybernetics and Systems: An International Journal*, 32:11–51.

Nehaniv, C. L. and Dautenhahn, K. (2002). The correspondence problem. In Dautenhahn, K. and Nehaniv, C. L., editors, *Imitation in Animals and Artifacts*, Complex Adaptive Systems, chapter 2, pages 41–61. The MIT Press.

Nehaniv, C. L. and Rhodes, J. L. (1997). Krohn-rhodes theory, hierarchies and evolution. In Mirkin, B., McMorris, F. R., Roberts, F. S., and Rzhetsky, A., editors, *Mathematical Hierarchies and Biology*, volume 37 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 29–42. American Mathematical Society.

Newell, A. and Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. In Anderson, J. R., editor, *Cognitive Skills and Their Acquisition*, chapter 1, pages 1–56. Lawrence Erlbaum Associates.

Nguyen, N. T., Bui, H. H., Venkatesh, S., and West, G. (2003). Recognising and monitoring high-level behaviours in complex spatial environments. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 620–625.

Nicolescu, M. N. and Matarić, M. J. (2001). Learning and interacting in human-robot domains. *IEEE Transactions in Systems, Man and Cybernetics, Part A: Systems and Humans*, 31(5):419–430.

Nicolescu, M. N. and Matarić, M. J. (2002). A hierarchical architecture for behavior-based robots. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, Part 1*, pages 227–233. ACM Press New York, NY, USA.

Nicolescu, M. N. and Matarić, M. J. (2003). Natural methods for robot task learning: instructive demonstrations, generalization and practice. In *AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 241–248, New York, NY, USA. ACM Press.

Nicolescu, M. N. and Matarić, M. J. (2007). Task learning through imitation and human-robot interaction. In Nehaniv, C. L. and Dautenhahn, K., editors, *Imitation and Social Learning in Robots, Humans and Animals*, chapter 19, pages 407–424. Cambridge University Press, first edition.

Nuxoll, A. M. and Laird, J. E. (2007). Extending Cognitive Architecture with Episodic Memory. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*.

Öhman, A. and Mineka, S. (2001). Fears, phobias, and preparedness: Toward an evolved module of fear and fear learning. *Psychological review*, 108(3):483–522.

Oliver, N., Garg, A., and Horvitz, E. (2004). Layered representations for learning and inferring office activity from multiple sensory channels. *Computer Vision and Image Understanding*, 96(2):163–180.

Omlin, C. and Giles, C. (1996). Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52.

Partington, S. and Bryson, J. (2005). The Behavior Oriented Design of an Unreal Tournament character. In *Proceedings of the Fifth International Working Conference on Intelligent Virtual Agents*, pages 466–477. Springer.

Pashler, H. (1998). *The Psychology of Attention*. MIT Press.

Paulos, E. and Canny, J. (2001). Social tele-embodiment: Understanding presence. *Autonomous Robots*, 11(1):87–95.

Pepperberg, I. (1994). Vocal Learning in Grey Parrots (Psittacus erithacus): Effects of Social Interaction, Reference, and Context. *The Auk*, 111(2):300–313.

Pfeifer, R. and Scheier, C. (1999). *Understanding Intelligence*. MIT Press.

Pylyshyn, Z. and Storm, R. (1988). Tracking multiple independent targets: Evidence for a parallel tracking mechanism. *Spatial Vision*, 3(3):179–197.

Quick, T., Dautenhahn, K., Nehaniv, C., and Roberts, G. (1999). The essence of embodiment: A framework for understanding and exploiting structural coupling between system and environment. In Dubois, D. M., editor, *Proceedings of the Third International Conference on Computing Anticipatory Systems (CASYS'99), Symposium 4 on Anticipatory Control and Robotic Systems*, pages 649–660, Liège, Belgium.

Quinlan, J. R. (1992). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ramachandran, D. and Amir, E. (2007). Bayesian inverse reinforcement learning. In Veloso, M. M., editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI '07)*, volume 1, pages 2586–2591, Hyderabad, India. IJCAI, AAAI Press.

Rankin, C. (2004). Invertebrate Learning: What Can't a Worm Learn? *Current Biology*, 14(15):617–618.

Rao, R., Shon, A., and Meltzoff, A. (2007). A Bayesian Model of Imitation in Infants and Robots. In Nehaniv, C. L. and Dautenhahn, K., editors, *Imitation and Social Learning in Robots, Humans and Animals*, chapter 11, pages 217–247. Cambridge University Press, first edition.

Ratnieks, F. L. W. and Anderson, C. (1999). Task partitioning in insect societies. *Insectes Sociaux*, 46(2):95–108.

Rennie, J. and McCallum, A. (1999). Using reinforcement learning to spider the web efficiently. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 335–343.

Rensink, R. A. (2000). The dynamic representation of scenes. *Visual Cognition*, 7:17–42.

Riegler, A. (2002). When is a cognitive system embodied? *Cognitive Systems Research*, 3(3):339–348.

Rieke, F., Warland, D., de Ruyter van Steveninck, R., and Bialek, W. (1999). *Spikes: exploring the neural code*. MIT Press, Cambridge, MA, USA.

Rizzolatti, G., Fogassi, L., and Gallese, V. (2000). Cortical mechanisms subserving object grasping and action recognition: A new view on the cortical motor functions. In Gazzaniga, M. S., editor, *The New Cognitive Neurosciences*, chapter 38, pages 538–552. MIT Press, Cambridge, MA, second edition.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326.

Roy, D. K. (1999). *Learning from Sights and Sounds: A Computational Model*. PhD thesis, MIT, Media Laboratory.

Roy, D. K. and Pentland, A. P. (2002). Learning words from sights and sounds: a computational model. *Cognitive Science*, 26:113–146.

Russell, S. (1998). Learning agents for uncertain environments (extended abstract). In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, New York, NY, USA. ACM Press.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition.

Sammut, C., Hurst, S., Kedzier, D., and Michie, D. (1992). Learning to fly. In Sleeman, D. and Edwards, P., editors, *Proceedings of the Ninth International Conference on Machine Learning (ML '92)*, pages 385–393, San Mateo, CA. Morgan Kaufmann.

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242.

Schaal, S. and Atkeson, C. G. (1994). Robot juggling: An implementation of memory-based learning. *Control Systems Magazine*, 14(1):57–71.

Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London, B*, 358(1431):537–547.

Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. (2004). Learning movement primitives. In *International Symposium on Robotics Research (ISRR2003)*, Springer Tracts in Advanced Robotics, Ciena, Italy. Springer.

Schlag, K. (1998). Why Imitate, and If So, How? A Boundedly Rational Approach to Multi-armed Bandits. *Journal of Economic Theory*, 78(1):130–156.

Schwartz, E., Greve, D., and Bonmassar, G. (1995). Space-variant active vision: definition, overview and examples. *Neural Networks*, 8(7-8):1297–1308.

Sloman, A. and Chappell, J. (2005). The altricial-precocial spectrum for robots. In *Proceedings IJCAI'05*, pages 1187–1192, Edinburgh.

Sommerville, I. (2006). *Software Engineering*. Addison Wesley Publishing Company.

Stein, L. (1994). Imagination and situated cognition. *Journal of Experimental & Theoretical Artificial Intelligence*, 6(4):393–407.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press (Bradford Book), Cambridge, Mass.

Teyler, T. J. and Discenna, P. (1986). The hippocampal memory indexing theory. *Behavioral Neuroscience*, 100:147–154.

Thiessen, E. D. and Saffran, J. R. (2003). When cues collide: Statistical and stress cues in infant word segmentation. *Developmental Psychology*, 39:706–716.

Thórisson, K. R. (1999). A mind model for multimodal communicative creatures & humanoids. *International Journal of Applied Artificial Intelligence*, 13(4/5):519–538.

Thrun, S. and Mitchell, T. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, 15:25–46.

Thurau, C. and Bauckhage, C. (2005). Towards Manifold Learning for Gamebot Behavior Modeling. In *Proceedings of the International Conference on Advances in Computer Entertainment Technolog (ACE)*, pages 446–449, Valencia, Spain. ACM SIGCHI.

Thurau, C., Bauckhage, C., and Sagerer, G. (2004a). Imitation learning at all levels of game-AI. In *Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education*, pages 402–408.

Thurau, C., Bauckhage, C., and Sagerer, G. (2004b). Learning human-like movement behavior for computer games. In *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB'04)*.

Thurau, C., Bauckhage, C., and Sagerer, G. (2004c). Synthesizing Movements for Computer Game Characters. In Rasmussen, C. E., Bülthoff, H. H., Giese, M. A., and Schölkopf, B., editors, *Proceedings of the 26th Pattern Recognition*

Symposium (DAGM '04), volume 3175 of *Lecture Notes in Computer Science*, pages 179–186, Heidelberg, Germany. Springer-Verlag.

Thurau, C., Paczian, T., and Bauckhage, C. (2005). Is Bayesian Imitation Learning the Route to Believable Gamebots? In *Proceedings of GAME-ON North America*, pages 3–9.

Tsotsos, J. K. (1990). Analyzing Vision at the Complexity Level. *Behavioral and Brain Sciences*, 13(3):423–445.

Tulving, E. (1972). Episodic and semantic memory. *Organization of memory*, pages 381–403.

Tulving, E. (1983). *Elements of episodic memory*. Oxford University Press New York.

Tulving, E. (2001). Episodic memory and common sense: how far apart? *Philosophical Transactions of the Royal Society B: Biological Sciences*, 356(1413):1505–1515.

Tyrrell, T. (1993). *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh. Centre for Cognitive Science.

Ueda, H., Yoshimori, T., Takahashi, K., and Miyahara, T. (2004). Categorization of continuous numeric percepts for reinforcement learning. In *Proceedings of the IEEE Region 10 Conference (TENCON 2004)*, volume B, pages 290–293, Chiang Mai, Thailand.

Valve (2004). *Half-Life 2*. Sierra Entertainment, Inc.

Vijayakumar, S., D'Souza, A., and Schaal, S. (2005). Incremental Online Learning in High Dimensions. *Neural Computation*, 17(12):2602–2634.

von Stein, A. and Sarnthein, J. (2000). Different frequencies for different scales of cortical integration: From local gamma to long range alpha-theta synchronization. *International Journal of Psychophysiology*, 38(301–313).

Wang, Y. and Laird, J. E. (2007). The Importance of Action History in Decision Making and Reinforcement Learning. In *Proceedings of the Eighth International Conference on Cognitive Modeling*, Ann Arbor, MI.

Watanabe, S. and Huber, L. (2007). Animal logics: Decisions in the absence of human language. *Animal Cognition*, 9(4):235–245.

Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Whiten, A. and Ham, R. (1992). On the nature and evolution of imitation in the animal kingdom: Reappraisal of a century of research. *Advances in the Study of Behaviour*, 21:239–83.

Witten, I. (1977). An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34(4):286–295.

Wood, M. A. (2006). Social learning and the brain. In Kovacs, T. and Marshall, J. A. R., editors, *Proceedings of AISB'06: Adaption in Artificial and Biological Systems*, volume 2, pages 64–67.

Wood, M. A. and Bryson, J. J. (2007a). Representations for action selection learning from real-time observation of task experts. In Veloso, M. M., editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI '07)*, volume 1, pages 641–646, Hyderabad, India. IJCAI, AAAI Press.

Wood, M. A. and Bryson, J. J. (2007b). Skill acquisition through program-level imitation in a real-time domain. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(2):272–285.

Wood, M. A., Leong, J. C. S., and Bryson, J. J. (2004). ACT-R is almost a model of primate task learning: Experiments in modelling transitive inference. In Forbus, K., Gentner, D., and Regier, T., editors, *Proceedings of the Twenty-Sixth Annual Conference of the Cognitive Science Society*, pages 1470 – 1475, Chicago, Illinois, USA. Cognitive Science Society, Lawrence Erlbaum Associates, Inc.

Zentall, T. R. (2001). Imitation in animals: Evidence, function, and mechanisms. *Cybernetics & Systems*, 32(1):53–96.

Zettlemoyer, L. S., Pasula, H., and Kaelbling, L. P. (2005). Learning planning rules in noisy stochastic worlds. In Veloso, M. M. and Kambhampati, S., editors, *AAAI*, pages 911–918, Pittsburgh, PA. AAAI Press.

Ziemke, T. (2001). Disentangling notions of embodiment. In *Workshop on Developmental Embodied Cognition (DECO-2001)*, pages 4–8, Edinburgh, UK.

Ziemke, T. (2003). What's that thing called embodiment? In Alterman, R. and Kirsh, D., editors, *Proceedings of the Twenty-Fifth Annual Meeting of the Cognitive Science Society*, pages 1305–1310, Boston, Massachusetts, USA. Cognitive Science Society.